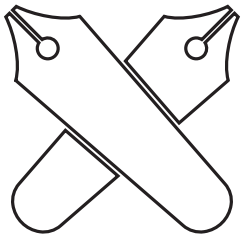
Doctoral Dissertation – Academic Year 2018

Towards Affordable Urban Computing through Deep Learning



A dissertation for the degree of Ph.D.
in Media and Governance

Graduate School of Media and Governance
Keio University

Makoto Kawano

Abstract of Doctoral Thesis Academic Year 2018

Towards Affordable Urban Computing through Deep Learning

Abstract

Thanks to urban computing, it becomes possible to make the city much smarter. However, it is still difficult to understand them because such situations are affected by many complex factors, spatial-temporal dependency, and other factors. While deep learning can be deal with the complex situation through a lot of high-quality data is required, it is not sufficient in terms of spatial-temporal aspect. Even if we collect big urban data, though mixing the urban data is important, the required knowledge about it is high-level.

Therefore, we propose a new framework affordable urban computing to facilitate understanding urban status by collecting and utilizing the enormous amount of urban data from limited existing urban resources. Since the city is the physical space, the texts written by people is the generic sensors to represent various status. Simultaneously, the image cameras are also very effective and very generic sensors that capture various status. With these sensors which have already existed in the city, we can exploit them to understand the city much deeper by using deep neural networks and edge devices. However, due to the nature of human activity, the distribution of the data is skewed and biased. Thus, we propose GeospaceMapping, which enables to estimate the spatial information represented in the data implicitly. In our experiment, we collected actual geotagged tweets and showed GeospaceMapping can estimate the prefecture. Then, we focus on dashboard cameras of garbage trucks which travel all around the city every day for their works film whole around the city. We propose an edge device CityInspector, which posses a function of the dashboard camera and road markings inspection and evaluate it in the experiment. Finally, we design and implement CityFlow exploiting GUI so that users can efficiently develop the application intuitively. We validate the effectiveness of CityFlow by implementing GeospaceMapping and CityInsepector.

Keywords
Urban Computing, Deep Learning, Edge Computing, Computer Vision, Natual Language Processing, Data Mining

Keio University Graduate School of Media and Governance
Makoto Kawano

博士論文要旨 2018 年度 (平成 30 年度)

深層学習によるアフォーダブル アーバンコンピューティング実現に向けた研究

論文要旨

現在の日本をはじめ，世界ではインフラ老朽化や人口減少などの問題を解決するために，IT 技術を用いたアーバン・コンピューティングによる課題解決が注目されている．一方で，都市状況理解は，様々な従属的な要因が複雑に絡み合っているため，容易ではない．そこで，都市の状況理解を促進する技術として，深層学習があげられる．深層学習応用には大量の良質なデータが必要となるが，時空間的に網羅したデータを集められていない．さらに都市データの種類は豊富であり，複数種類のデータ利活用が有効的であるとされているが，組み合わせ方などについては明らかになっていない．

本研究では，大量の良質な都市データを現在の都市の限られた資源から収集し，これらの包括的な利活用による都市理解を促進するアフォーダブル・アーバン・コンピューティングを提案する．対象としている都市は現実における物理空間である．都市の複雑な状況を表現する文章データを出力する市民をはじめとした人間センサと，ドライブレコーダなど都市の状況を撮影可能なカメラなどの汎用的な画像センサが存在しており，これらの利活用は有効である．しかしながら，人々の行動の性質から，人間センサから取得可能なデータには時空間的偏りが生じてしまう．本研究では，深層学習を用いて文章や画像が潜在的に指し示す空間情報を推定する GeospaceMapping を提案する．実験では，実際に収集した位置情報付きツイートを用い，投稿されたテキストと画像の組み合わせから都道府県の推定ができることを検証した．さらに，カメラ映像は本来の目的が都市の状況理解のためではないため，有効活用されずに破棄されてしまっている．本研究ではゴミ清掃車のドライブレコーダに着目する．これは，日々の業務のために複数台が市内を網羅的に走行しており，都市における空間的な網羅性は高い．そこで，ドライブレコーダの機能を有し，道路標示の損傷検出を行うエッジデバイス CityInspector を提案し，実装および実験を行い，その有用性を検証した．最後に複数センサの組み合わせを GUI によって直感的に開発が可能となる開発環境 CityFlow の設計および実装を行った．そして，実際に CityInspector や GeospaceMapping を実装し，その有用性を示した．

本研究の貢献は，深層学習およびエッジデバイスの利活用と，これらを統合するインターフェースの提供により，都市の状況理解促進研究分野を開拓した点にある．

キーワード

アーバンコンピューティング，深層学習，エッジコンピューティング，画像認識，自然言語処理，データマイニング

慶應義塾大学大学院 政策・メディア研究科
河野 慎

Acknowledgments

I would like to express really my gratitude to my advisor, Professor Jin Nakazawa. Professor Nakazawa always adviced me to take action on my own and cheered me. This has never changed since I consult him about my course in graduate school.

I am grateful to members of the thesis committee. I express my gratitude to Professor Hideyuki Tokuda. Professor Tokuda was my former adviser and he always took time out of his busy schedule and advised me. After retiring the university and acceding to President of National Institute of Information and Communications Technology, he always took care of my research and cheer me. I am also indebted to Professor Yasushi Kiyoki for taking care of my research since I have met him at DEIM. I would like to thank Professor Yoshiyasu Takefuji for taking time whenever I asked him and encouraging me. I also would like to thank Professor Yutaka Matsuo for undertaking to join the committee despite the suddenness and giving me a lot of advice about the thesis.

I would like to express all of the members of our laboratory, especially to Dr. Tadashi Okoshi and Dr. Yin Chen for their advice and encouragement. Needless to say, I am very grateful to Dr. Takuro Yonezawa for his kindest advice and encouragement since I was an undergraduate and assigned the laboratory. I also thank to the member of the d-hacks members – especially Yuki Noguchi, Hirono Kawashima and Tomoki Tanimura who support group management and my research. Without their support, I would have not been able to enjoy the research life in the laboratory.

I cannot help thanking the member of the Matsuo Laboratory where is at the University of Tokyo. Especially, Yuya Soneoka always talked to me very enjoyable and Kaoru Nasuno is always good friends to me. Needless to say, Dr. Masahiro Suzuki always advises me about my research in terms of theoretical aspects. Of course, Dr. Yusuke Iwasawa always helped me whether business or personal and I always lean on him for advice. All my thankfulness to them cannot be expressed in this acknowledgments.

I would like to thank Yu Ohya of NTT, Inc. He always offered some advice about my research whenever I asked him. Thanks to his advice, finally I have accomplished to submit the paper to several domestic conferences in terms of methodological aspects. Those experience will help me rest of my research life.

I am also grateful to Maho Morimoto that she is always with me and encourage me

by taking me to the Disneyland, DisneySea and so on. Without her support, I would always have been depressed and I have not accomplished to write this thesis.

Last but not least, I would like to take this opportunity to express my sincere gratitude to my family – especially to my parents, Seiichi and Kumiko Kawano – for always supporting me. They always concerned me whenever I come back to home.

Dedicated this thesis to my late grandmother, Tokie Kawano.

February 17, 2019
Makoto Kawano

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

How can we improve our quality of life in a city? One of the key concepts of quality improvement is a *smart city* [8]. Deakin and AI Wear [9] defined the smart city with some points of view: using information and communication technologies to enhance the quality, performance, and interactivity of the city services. There are a lot of cities to realize the smart city such as Singapore, Amsterdam, China and so on. For example, in Spain, SmartSantander [1] is one of the famous smart city projects. In Santander, a lot of sensors are embedded in anywhere such as each parking spots, which publish the data whether the parking is full or not. Needless to say, Japan also tries to realize the smart city. For instance, Sapporo city conducts the project 'Sapporo Smart City Project" from 2015, which is supported by the Ministry of Land, Infrastructure, Transport, and Tourism and the Ministry of Internal Affairs and Communications. This project aims to save energy and electricity for their environment.

One of the research fields to realize the smart city is an urban computing [10][4]. Urban computing is motivated to address the big issues in big cities, such as traffic congestion, energy consumption, and pollution, by using big data. The goal is creating solutions that improve the city environment, citizens' life quality and city operation. For example, in order to exploit the root cause of air pollution, Zheng et al. [11] have studied the correlation between air quality and traffic flows (and points of interest). Since air pollution would violate the health of citizens, this study is very significant for the city.

In the meantime, deep learning is widely used in various research fields. Deep learning is one of the machine learning methods and it is spread rapidly starting by winning in the ImageNet Large Scale Visual Recognition Competition 2012 [12]. Thanks to their performance of image recognition or natural language processing, deep learning is applied to the industrial field, such as autonomous cars, manufacturing, and robotics. Thus, it can be expected to use deep learning methods to understand the city much deeper.

1.2 Research Goal

The goal of this research is that to create a new framework of urban computing throughout deep learning, since understanding the status of the city is quite difficult due to

[1] http://www.smartsantander.eu/

their complexity. To utilize deep learning effectively, the enormous amount and the high quality of data are required. Our proposed new framework, called *affordable urban computing* tackled to provide the urban data so that urban computing through deep learning is made to afford.

Firstly, in terms of human as a sensor, so-called participatory sensing is a good way to get the urban data dynamically. However, people are very sensitive about privacy concerns so that the reports become sparse. For instance, Europe enforced the law about the privacy concerns, so-called the General Data Protection Regulation (GDPR), which protects the privacy for all individuals within the European Union and European Economic. While Zheng et.al [13] has distributed the devices to volunteers in order to gather the GPS data for their experiments, the GPS data has the potential to reveal the daily life of the volunteers. Moreover, it is not realistic to distribute the devices to all citizens. In short, it is possible to conduct data analysis for human life quality, yet it is difficult to conduct in daily sustainably.

The second problem about the urban environments is that conventional sensors are very useful to capture the target phenomenon such as temperatures or humidity as long as the sensors become broken. However, because the city is a continuous space, those sensors cannot cover all around the city. Therefore, many of previous studies have attempted to interpolate the data not only missing but also nothing in the first time. Of course, this is one of the solutions to deal with capturing the urban continuous space, but it has a limitation. Indeed, we need to explore the way to capturing the urban continuous space in full as much as possible.

The third problem of city operation systems is not only the cost but also the technical domain knowledge of local governments. When the problems mentioned above are solved with reference to low-cost sensing, we have to think of manipulators for those sensing technologies. In general, the manipulator is assumed to be a company or a university because they have technical knowledge and skills. If the manipulators are the company, although it can be trusted the quality of their jobs, it takes a lot of fee for the contract. In the other hand, while the fee of the contract with the university is much lower than that of the company, it is difficult to work sustainably. Therefore it is preferable to manage the city operation systems within the local governments. In addition, the targets to be operated in the city environments is the data and the devices of which the variables are full. Thus, we need to treat them easily.

1.3 Thesis Statement

In this thesis, we first overview the smart city cases taken place in the world and also the current status of Japan to clarify the problems that past urban computing is valuable but not realistic to be pervasive. The main body of this thesis is a proposition of new framework, affordable urban computing, for facilitating local governments to introduce urban computing to their city. The contributions of this thesis are summarized as follows:

- Define the problem in introducing deep learning for practicing urban computing in the real world.
- Proposing the Affordable Urban Computing that enables to increase the amount and change the quality of urban data to make urban computing affordable.
- Proposing the geospace mapping method that attaches spatial information to non-geotagged information so that we can regard them as location-based information.
- Proposing the city condition sensing method that augments the daily works of local governments and enables them to grasp the current city status sustainably.
- Designing and implementing a development environment that enables local governments to develop and manage the application with a variety of data and devices.

With affordable urban computing, the amount and the quality of urban data become well so that we can use the urban data for deep learning approaches. With affordable urban computing, we can develop and manage those urban data and devices easily so that not only the experts but also beginners can do as same.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 presents the current city status and progress towards being smart cities with urban computing. Then, we will describe our three technologies and the problems of existing research in Chapter 3 In Chapter 4, we will describe GeospaceMapping, which enables us to understand where the tweets mentioned implicitly by using deep neural networks. In the next Chapter 5, we will

explain one of the practices of urban recycled sensing, we called CityInspector, which can inspect the condition of roads while the city officers carry out everyday tasks, such as garbage collecting and go on patrol. In Chapter 6, we will introduce the new development environment, CityFlow. Chapter 7 describes related work to reveal the contribution and the position of this thesis. Finally, we conclude this thesis in Chapter 8.

Chapter 2

Background: Towards Being Smart City

2.1 Introduction

In this chapter, we represent the background of our research: smart cities, urban computing, and deep learning. Firstly, we introduce the definition of smart cities with several global/domestic proof of concept. Then, we describe urban computing which this thesis focuses on. Lastly, we present deep learning, which is a key technology that is mainly used in this thesis.

2.2 Smart City

Smart City is a novel city that manages its assets and resources efficiently by using information and communication technologies (ICTs). With the combination of technology of big data processing due to the internet (i.e. web) being widely spread, and the internet of things (IoT) that the various kind of furniture and embedded computer (such as Raspberry Pi) are connected to the internet, it is capable of data being collected from all over the city. Utilizing the data collected from the city enables the city to monitor and manage itself well-efficiently. For example, monitoring traffic and transportation flows and managing power plants and waterworks bureau. Though there are many cities that practices to make itself smarter, the definition of "smart city" is slightly different in the viewpoint of their practices. In this thesis, we follow the explanation and definition of the smart city given by Deakin and AI Wear[9] that the smart city is composed of followed four components:

- *the application of a wide range of electronic and digital technologies to communities and cities,*
- *the use of information technologies to transform life and work within a region,*
- *the embedding of such ICTs in the city,*
- *the territorialization of such practices in a way that bring ICTs and people together, so as to enhance the innovation, learning, knowledge, and problem solving which they offer.* – Deakin and AI Wear [9]

Note that these four components are firstly mentioned by Hollands [8] that he drew attention to the work of Komnionos [14][15] on the intelligent city, and then Deakin and AI Wear defined with appending their interpretation; the smart city should be introduced not only to solve each separated problem but also make the local communities

better. That is, the smart city is that the city, especially local governments, and the citizens join hands for solving the city problems by using ICTs.

On the other hands, in Japan, the smart city is mentioned in the context of "Society 5.0". Society 5.0 is the model that aims to create a novel value from a conventional information society by the system fusing cyber-space and physical-space. In current society, so-called "Society 4.0", the human resource cost also has been higher due to decreasing birthrate and aging population, for instance. In order to alleviate the cost and solve those aging population problems, it is important to realize the society to be smart by using ICTs.

Given the practices of smart cities that are conducted in the world, there are mainly three technologies that are used as follows:

- Internet of Things (IoT) Service.
  In terms of IoT, the sensors that can connect to the Internet are installed in the city such as parking and rental cycle. The data from the sensors are used for monitoring usage of service E.g. Manchester [16], San Leandro [17], Shanghai [18], Singapore [19].

- Open Data Platform.
  Thanks to the ICTs spread widely, the variety of data is able to be obtained. The challenge with reference to this is data management. Many cities construct own platforms that can treat various data format as the same one and offer it as open data. E.g. Dublin [20], Milton Keynes [21].

- Operating System.
  In terms of operating systems, some cities introduce the e-service that realizes paper-less or cash-less to reduce the cost and manage them in the digital space Moreover, there is some trial to introduce an electric car to be clean for the environments. E.g. Barcelona [22], Columbus [23], Dubai [24], Stockholm [25], NewYork [26].

2.2.1 International Testbed Studies

Firstly, we introduce the two case studies of smart cities that are conducted outside of Japan. Note that there are still many cases as not mentioned in this thesis, we focus on the cases that conduct the projects in terms of various aspects.

Figure 2.1: The sensors that are used in Smart Santander project [1].

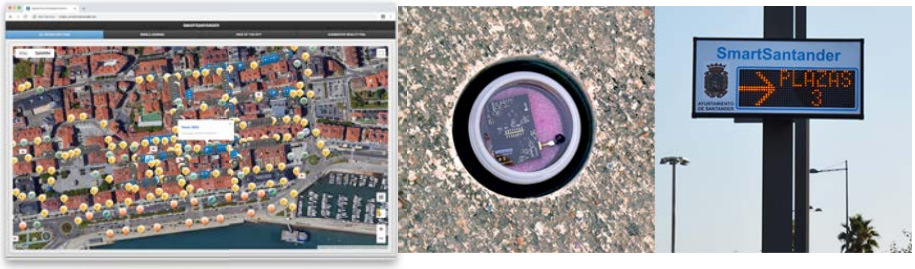


Figure 2.2: The parking management system of Smart Santander [1]. (Left) the web screen of appliction (`http://maps.smartsantander.eu/`). (Right) The actual sensor embedded in parking and the sighboard. These images are refered from `http://www.libelium.com/smart_santander_parking_smart_city/`.

Smart Santander

One of the famous smart city projects is "Smart Santander" [27], which was funded by the 7th Framework Programme, EU. Eight EU countries (i.e Spain, France, Germany and so on) and 15 organization, such as Telefonica from Spain, participate in this project. Several cities are selected to be a test-bed of smart city and more than 20 k of sensors are installed in the city.

In particular, in Santander City, 2000 of sensors that 650 targeted to service provision (temperature, noise level, light intensity, CO) are installed on lamp posts and 400 of sensors are buried in the asphalt as depicted in Figure 2.1. They have developed several applications by using those sensors. For instance, they have developed parking application controlling traffic: showing parking availability with special electronic signs/web interfaces Figure 2.2.

Amsterdam Smart City

Another big smart city project is "Amsterdam Smart City" [28]. Amsterdam Smart City is the platform to tackle urban issues. The distinguishing feature of this project is that those projects taken place in Amsterdam is not decided by a top-down approach such as other cities do, but applied from public authorities, businesses and citizens through the "Amsterdam Circular Challenge" held on every year. There are more than 270 applied projects, which follow within seven themes: digital city, energy, mobility, circular city, governance & education, citizens & living, and smart city academy.

For example, the aforementioned technologies, open data platform "City Data" [2] has been developed. The City Data contains big data collections including all addresses of Amsterdam, topographical data, cadastral data and so on and they offer the portal site (Figure 2.3) that the data can be downloaded with any kind of data format (e.g. csv, json) Moreover, in terms of IoT, there is a project called LoRa Smart Lighting + Lightwell. The smart lighting enables local governments to remote control the brightness of streets lights via LoRa network.

2.2.2 Domestic Case as Testbed Studies

Simultaneously, we present the case studies of smart cities that are taken place in Japan. These cases are mainly sponsored by the Government of Japan: the Ministry of Land, Infrastructure, Transport and Tourism, and the Ministry of Internal Affairs and Communications.

Sapporo City

Sapporo city works on a smart city project from March 2017 with general incorporated foundation, the foundation of Sapporo industrial promotion. This project aims to utilize open data by constructing the platform of ICT utilization, *DATA-SMART CITY SAPPORO* [3]. The DATA-SMART CITY SAPPORO lets company and university to use open data and citizens to become familiar with open data. For example, the DATA-SMART CITY SAPPORO provides the contents (e.g. flu or the information about the winter road condition) for citizens and the result of analyzing collected open data for company and university, respectively. The actual image of DATA-SMART CITY SAPPORO is shown in Figure 2.4. Mainly, there are two functions in the DATA-SMART CITY SAPPORO as follows:

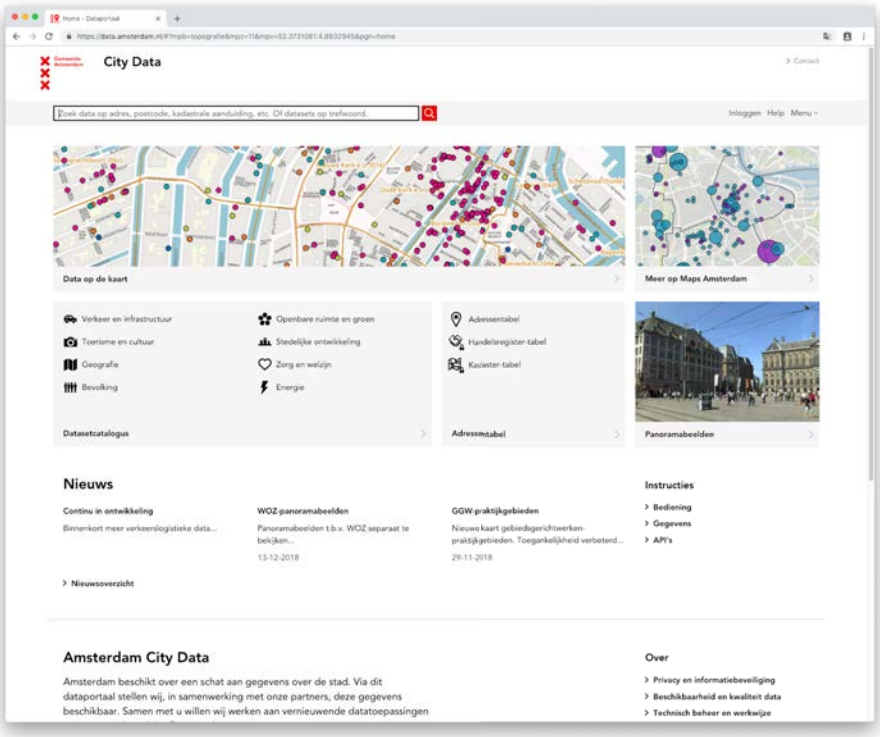


Figure 2.3: CityData [2]. The screeshot of portal site(`https://data.amsterdam.nl/`).

- Data Catalog.
  This function provides the data classified into 12 groups (e.g. disaster prevention, population, mobility): the information of shelters and the future estimated population.

- Dash Board.
  This funtion displays the information of events and the status of flu epidemics by using maps and graphs of the data analyzed results.

Moreover, they conduct some studies as proof of concept. For example, in order to find the spot where snow is to be removed, they merge and use the data gathered taxi, bus, and the garbage trucks, and also the data from citizens through participatory sensing.

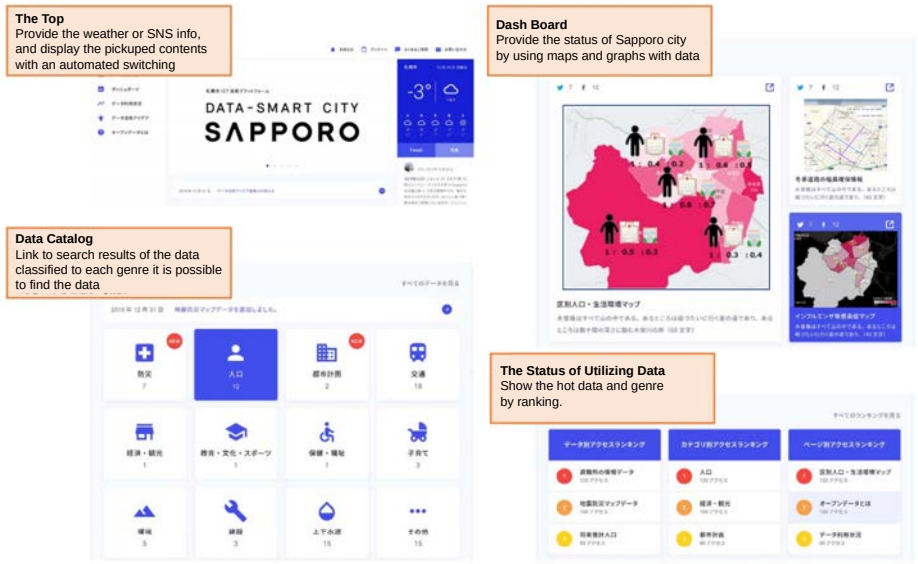


Figure 2.4: DATA-SMART CITY SUPPORO [3]. The Japanese of original figure is partly changed to English.

Takamatsu City

Another example of smart cities in Japan is studied in Takamatsu City, Kagawa Prefecture. They use FIWARE to build the common IoT platform. Then, they use this platform for four regions of city work: disaster prevention, sightseeing, welfare, and transportation. For disaster prevention, they install water level sensors and tide level sensors into rivers so that the city officers can grasp the status of the rivers. When the sensors detect anomaly status, the city officers announce the anomaly and guide the evacuation routes via the smartphone application.

In the sightseeing region, in order to discover and understand the hot spot for foreign tourists, Takamatsu City installs GPS loggers to rental-cycles which they provide. The GPS loggers offer the geolocation data where the tourists start and end the riding bicycles. With the geolocation data, Takamatsu City can understand the transit time, routes, and the range of activity so that they can manage tourist attractions.

For welfare region, Takamatsu City collaborates with technical college and company to develop the wearable best which multiple sensors are equipped with. This wearable best is used for watching old people to detect fall and know where they are.

Lastly, in the transportation region, knowing the occurrence spots of a near-miss

event is very significant for the safety of citizens. Moreover, understanding the factor of a near-miss event is also significant too. Therefore, in order to gather the data of the places of near-miss events, Takamatsu City collects the videos of dashcams and conduct image processing to extract the factors.

2.3 Urban Computing

One of the research fields that tackle urban issues from a different perspective is "urban computing" [10]. Urban computing is a new paradigm shift from ubiquitous computing [29], that is, a lot of devices are installed everywhere in urban space and the data about an urban environment from them is gathered for any application. In this thesis, we adopt the definition of urban computing which is given by Zheng et.al [4]:

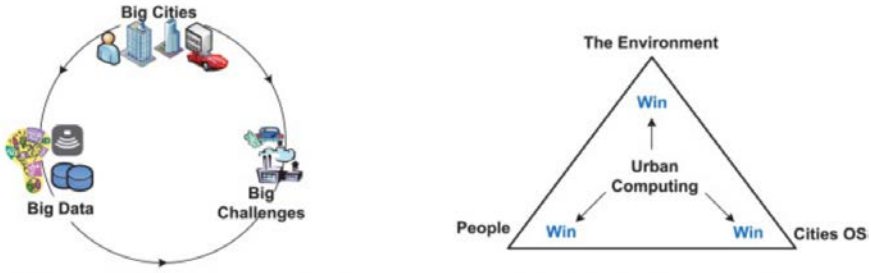
> *"Urban computing is a process of acquisition, integration, and analysis of big and heterogeneous data generated by diverse sources in urban spaces, such as sensors, devices, vehicles, buildings, and humans, to tackle the major issues that cities face."*

Given this definition, though urban computing seems to be similar to conventional sensing networks, the most different viewpoint is urban computing assumes that there is various kind of devices, data (formats), and human interaction.

The motivation of urban computing is in order to tackle big challenges that emerge in the modernized big city lives such as air pollution, energy consumption, and traffic congestion, using big data that is produced by sensing technologies and large computing infrastructures. This motivation is illustrated in Figure 2.5. With this motivation, the goal of urban computing becomes as follows:

> *"Urban computing connects unobtrusive and ubiquitous sensing technologies, advanced data management and analytic models, and novel visualization methods to create win-win-win solutions that improve the urban environment, human life quality, and city operation systems...."* [4]

which Zheng et.al [4] has mentioned. We consider this point as very significant because this point is fit for the definition of smart cities. Thus, practicing urban computing makes the city smarter.



(a) Motivation: Big cities, data and challenges (b) Goal of urban computing

Figure 2.5: Motivation and goal of urban computing [4].

2.3.1 Examples of Urban Computing

With reference to Zheng et al. [4], there are several themes that urban computing tackles. In this section, we briefly introduce the examples of urban computing in the three aspects according to Figure 2.5: environment, human, CityOS.

Environment

Recently, we have observed that the air is polluted by PM2.5 or pollen spread especially in Japan. Simultaneously, there is noise pollution in a number of countries, such as the United States, the United Kingdom, and Germany. That is, it is important for citizens and cities to monitor and protect the environment. In terms of environmental monitoring, there are some works to estimate the urban air quality based on various sensors [30][31]. Another example of noise pollution, Rana et al. [32] have used a smartphone to measure the sound of noise and then map the result to the systems. Another work related to environments is energy consumption. For example, Zhang et al. [33] have attempted to detect the refueling behavior from GPS trajectories collected from taxies in Beijing.

Human

The challenges about the human in the city are that what kind of people is live in and whether the city is safe/secure. There are a lot of studies measuring the similarity of people by using the data of social media services or mobility data [34][35]. In particular, location-based social networks, which location information is attached to existing contents such as text and images, are very useful for similarity measuring,

because individuals' location history in the real-world implies their interests[36] and behaviors. Therefore, the similarity between users by comparing with their location histories can be used for user recommendation [37] or community discovery [38].

On the other hand, people do not want to live in a city where crimes often occur or that is not safe/secure. There are some works that predict the place and the time is at high risk for future crimes [39] or study the relationship between urban form and gang activity [40].

City Operation System

The last concept is the city operation system assuming that mainly local governments use the organization managing the infrastructures such as urban transportation (i.e. buses, taxies, trains, and subways). For local governments, effective planning is very significant to build a smart city. In order to formulate urban planning, a local government is required to grasp various factors, such as traffic flow with road network structures. Zheng et al. [41] have estimated the underlying problems in Beijing's transportation network by analyzing the GPS trajectories from the taxies over 3 years and found that a newly launched subway line solved the underlying problem that traffic flow is slow. Another research related to urban planning is Yuan et al. [42]. Yuan et al. have proposed a framework that discovers regions of differnet functions in a city such as the educational and scientific, or commercial areas in Beijing. On the other hand, for managing transportation, there are some works to improve the taxi services [43] and bus services [44].

2.4 Deep Learning

In the meanwhile, deep learning, nowadays, has been well-known all over the world because of its capability. While deep learning is just one of machine learning methods, Goodfellow et al. [45] has said in his book that deep learning helps computers to understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts. The history of deep learning is so long that it is said that the concept of deep learning is first known as cybernetics in the 1940s – 1960s, then known as connectionism in the 1980s – 1990s, and currently beginning to be known as 'deep learning' in 2006. The most breaking news that deep learning has been widely known is that a convolutional neural network, which is one of the fa-

Figure 2.6: The example of Mask R-CNN [5]. This image is refered from `https://github.com/facebookresearch/Detectron`.

mous architecture of deep learning, won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for the first time and by a wide margin in 2012 [46].

With this breakthrough, deep learning is rapidly applied to the various scene, such as natural language processing [47], and speech recognition [48]. For example, in terms of image recognition, deep learning is able not only to classify given images into multi-class labels but also detect the same instance of objects in a given image [5] as depicted in Figure 2.6. Besides the image recognition, it has come up in the news in the context of NLP that the algorithm of Google Translation systems [49] is replaced with a deep neural network, whose architecture is so-called LSTMs [50].

In addition, extending deep learning to the domain of reinforcement learning achieves good results [51]. Reinforcement learning is that train an autonomous agent to perform a task by try and error without any explicit guidance. One of the famous results is AlphaGo that defeats a world champion at ‘Go‘ game [52]. Furthermore, reinforcement with deep learning has been applied to robotics [53]. The reason for the progress of deep learning is a technological innovation including GPUs and big data (e.g. from 1.2M [54] to 300M [55]).

To summarize, deep learning has a great potential to solve the real-world problem, yet the crowning achievements are in a limited domain. While there are some works that use deep learning for urban computing [56][57], it is not sufficient and we have to explore the usability of deep learning for urban computing.

2.5 Summary

This chapter represented the background research of this thesis. In order to improve the quality of citizens life, a lot of cities introduce IT technologies, such as IoT and big data

platforms, with the support from the government in the context of smart cities. One of the technologies that make a city smart is urban computing using various data to solve urban challenges. We describe some examples of smart cities and urban computing respectively. However, using these technologies are assumed to have an enormous budget and computer resource. In the next chapter, we describe the problems that occur when small cities attempt to introduce these technologies.

Chapter 3

Affordable Urban Computing

3.1 Introduction

In this chapter, we present a new computing paradigm called *affordable urban computing* which we propose in this thesis. To motivate our proposal, firstly, we explain the current status of Fujisawa City which has not been yet solved by conventional urban computing approach. Then, we define the term "affordable urban computing" with their requirements. Lastly, we discuss questions of this research to be solved in order to realize the concept.

3.2 Unsolved Urban Problems

In the conventional urban computing research field, aforementioned large scale applications, such as air pollution or traffic flows are focused because those cities where practice smart city projects are tends to be a big city. Of course, it is not only the applications for big cities but also for small (not big) cities, for example, Fujisawa City in Japan. In addition, there are a lot of problems which are mentioned in the previous chapter. For instance, the infrastructure of the city, such as road networks and tunnels, because the infrastructure has been often constructed in a period of high economic growth. Therefore, local governments are required to investigate the infrastructure whether it needs to be repaired or not, and if it is, plan when they repair the infrastructure. Similarly, local governments want to know how much garbages are taken out during one day. Moreover, they want to know how many people visit their cities because the number of people visits their sightseeing spot is very important for their urban planning.

In order to know the status of the cities, hiring people to observe and report the status is a simple, easy and quick way to begin. However, especially in Japan, the population become decreasing so that there is a prediction that the population of Japan will become much less than recent. Moreover, while a 'working' population will be decreased by 40% until 2065 [1] due to the population decreased, amount of work of local governments be increased rather than being unchanged because of the rate of aging people will glow. Therefore, it is not realistic to hire the people for understanding the cities.

With these above reasons, local governments also want to introduce urban com-

[1] Investigated by Mizuho Research Institute Ltd. `https://www.mizuho-ri.co.jp/publication/research/pdf/insight/pl170531.pdf`

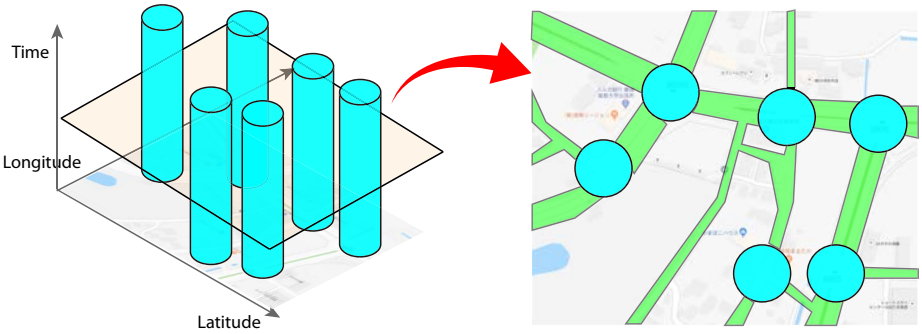


Figure 3.1: Coverage of conventional sensors installed in the city roads. If the sensors are installed in the intersection (blue areas), we cannot get the information about the green areas where are out of the areas that sensors can capture.

puting technologies in their cities to make it smarter, that is, they want to understand in more detail and with more precision how citizens live in their cities and how the environment of their cities is.

Nevertheless, the conventional urban computing such proposed by Zheng et al. [4] is not able to satisfy the requirements to solve aforementioned challenges, because the data which they have used in their studies is not much fine-grain in spatial-temporal aspects. Namely, as to road inspection, while Zheng et al have also mentioned that we do not have sensors on every road hence we should use human as a sensor instead, still, human as a sensors is not enough to capture 'all' damaged roads. One of the reasons that human as a sensor is not enough is that because the privacy of users who report a road damage may be violated, users dislike reporting. Another reason is that it is quite difficult for users to see all roads as depicted in Figure 3.1, the green area being out of the conventional sensors (blue areas).

In summary, we have to propose a new framework that can tackle such challenges that require affordable sensing in the context of spatial-temporal and the environment that enable to develop and manage the data from affordable sensing and the devices easily.

3.3 Affordable Urban Computing

In this section, we describe a new concept of affordable computing, which aims to solve the lack of data and development environments problems so that makes urban computer able to afford. First of all, we define the terms "affordable urban computing",

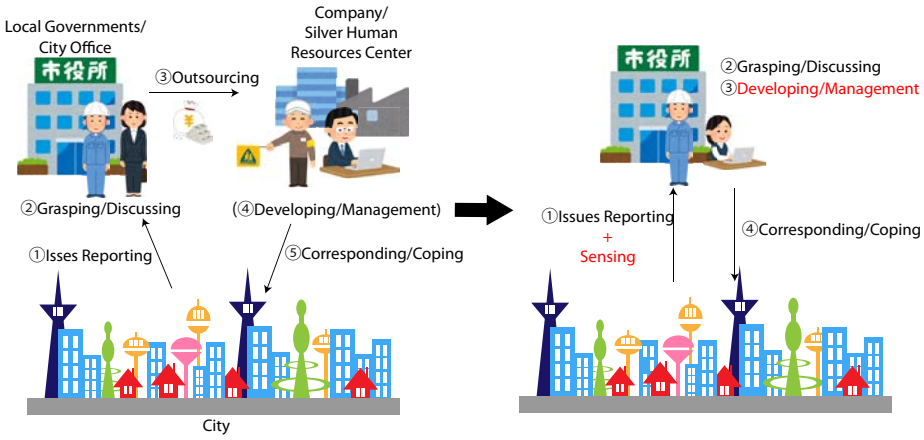


Figure 3.2: The illustration of before/after introducing affordable urban computing. Before introducing affordable urban computing, 1) when local governments get some issue reports, 2) they discuss the issues whether to be solved or not immediately. 3) If they decide to solve issues, they often outsource them to some company. 4) The company plan the approach to solve the issues 5) and solve them.

then we explain the requirements to realize the affordable urban computing.

3.3.1 Definition

Affordable urban computing is a new concept that aims at efficient operating of (complemental) data and context information by utilizing "things" existing in the city as sensors, in order to afford urban computing. A term of *city* or *urban* indicates a local public entity well-defined in the constitution. Therefore, in terms of a local public entity, we can also regard some 'town' or 'village' which is so-called 'rural area' as 'urban' or 'city', though the budget of towns or village is much fewer than that of cities. Affordable urban computing combines deep learning methodology and edge computing, which lets us operate the various modal data/information with low costs.

Figure 3.2 shows the process of solving urban challenges with/without introducing affordable urban computing. In the current city, when a local government gets some issue reports, they discuss the issue of whether to be addressed or not as a priority. As the local government decides to address the issue, they outsource it to a company if the local government cannot address the issue by itself. On the other hand, using sensing technologies and a development environment of affordable urban computing, the local government unable to address the issue by itself.

3.3.2 Information Types in Cities

In the city, there are enormous kinds of data. In this thesis, we categorize the data into two types: geospatial information and urban information. According to Article 2 of the Basic Act on the Advancement of Utilizing Geospatial Information, geospatial information is defined as:

- The term "Geospatial Information" as used in this Act means information consisting of either item (i) or a combination of items (i) and (ii):
- (i) information that represents the position of a specific point or area in geospace (including temporal information pertaining to said information, hereinafter referred to as "Positional Information"); and
- (ii) any information associated with the information in the preceding item of this Article.

Simultaneously, we also define urban information as the information that describes the status of anything about the city, such as images and text.

3.3.3 Requirements

The major goal of affordable urban computing is to enable the cities where is willing to monitor and understand their citizens' lives and the status of their city environment. Figure 3.3 illustrate a general framework of affordable urban computing. A part surrounded by red dash lines is our affordable urban computing, while except the part is out of the scope of this thesis. Indeed, a part of participatory sensing in a human as a sensor and conventional sensors are out of scope in our thesis because many research has studied in those technologies. Though, we introduce participatory sensing briefly in the next section. To realize affordable urban computing, we need technologies that fulfills the following two requirements.

- **Ecological Urban Sensing.**
  Urban computing requires a lot of data to conduct machine learning approaches and analysis, so-called big data. Of course, Zheng et al. [4] have mentioned they use the data being both geospatial information and urban information. However, in the city, there is still a lot of data that is not used not at all even though it is either geospatial information or urban information. For example, the images

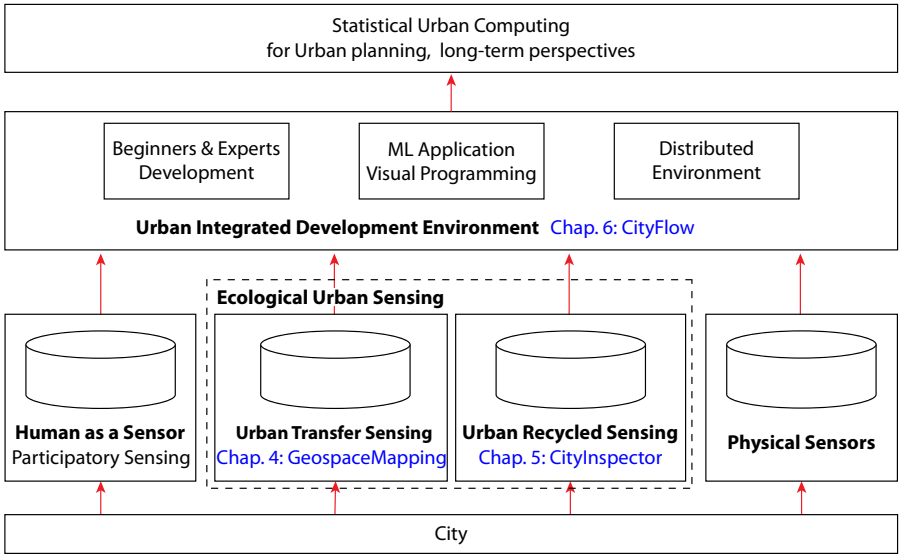


Figure 3.3: General framework of affordable urban computing.

filmed by cameras existing in the city, such as dashboard cameras or surveillance cameras. Otherwise, the web data or log data that does not contain geospatial information. Indeed, if we attach the geospatial information to urban information or vice versa, we can afford the data to urban computing so that we can enhance and facilitate urban computing. Nevertheless, the conventional way to utilize the data is post-processed after aggregating the data. This is very troublesome due to the difference in the format of data or the procedures. In order to remedy this problem, those cameras should be implemented as edge devices.

- **Unnecessity of Expert Knowlegdement.**
  The variety of urban data is enormous in the city and it is expected to use them as much as the performance of approaches such as deep learning is increased. However, it is difficult for us to know how can we mix the various data in appropriately. For example, with the data that users posted in social network service, which contains not only the text but also images or users' social network. What is the good way to combine these kinds of data? While, in general, this is a role for experts such as data scientists, there is merely no experts in local governments. Indeed, we need the environments that can examine the performance of the combination of data and the stretegies to combining the data.

- **Easy Development and Management.**
  Due to conducting affordable urban sensing, the enormous amount of data is generated in the city. In terms of efficiency, developers must be able to manage the data flow and utilize it easily. We consider that the number of application of affordable urban sensing might be large so that it is troublesome to develop every application of affordable urban sensing. With regard to affordable urban sensing application development, developers are required to consider the specification and contexts of devices which are deployed the application. For supporting, easy development and management environment need to be realized.

  Moreover, while a developer would be an expert of programming, networking and machine learning generally, we assume that the beginners such as the city officers of local government even citizens may develop the applications. Therefore, the development and management environment need to become much easier for beginners.

3.3.4 Relation to Urban Computing

Affordable urban computing is, of course, very similar to conventional urban computing proposed by Zheng et al. [4]. The main purpose of both concepts is the same that both aim to tackle urban challenges and utilize human as a sensor. However, the urban data is not sufficient for utilizing deep learning application. Even if the urban data is collected enough for deep learning, the way to mix the urban data for deep learning and to develop and manage the urban data with the devices are troublesome. Our affordable urban computing is a new framework to tackle these problems within the current limited resources (data and environments) and realize urban computing to be more affordable. While the difference between them is that our affordable urban computing attempts to increase a 'gradient' that obtained value per a cost more than vanilla urban computing as illustrated in Figure 3.4.

3.4 Problems for Realizing Urban Computing

3.4.1 Affording Data Problem

In order to realize urban computing, we focus on 'text' sensors and 'image' sensors; we utilize data of social network service and data of cameras installed in the city. Since the city is very complex physical space, these 'flexible' sensors are effective to capture the

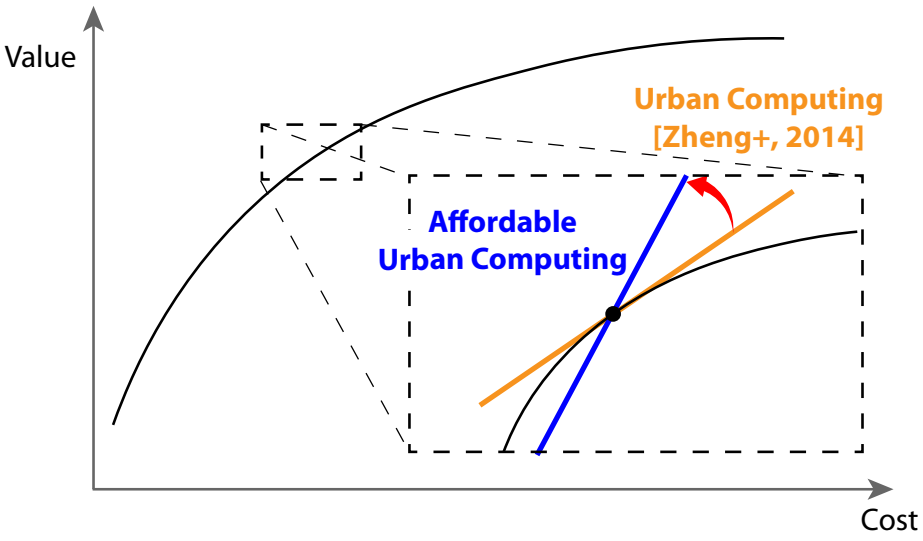


Figure 3.4: The illustration of the difference between our affordable urban computing and vanilla urban computing proposed by Zheng et al. [4].

status or the context of the city. However, the data from the flexible sensors are not able to be used straightforwardly due to the data which is also complexity. To remedy this problem, we adopt deep neural networks because those enable to extract the arbitrary information as the representation. However, there is no study that extracting urban information by using deep neural networks, as our best knowledge. Therefore, this is the problem which should be revealed whether it is possible or not.

Geospatial Information Attachment

Though it is able to monitor parking availability if the small cities enable to purchase sensors such as cameras or the sensors embedded in the roads as Smart Santander [1], it might be difficult to install a sufficient number of sensors without enormous supports such as national foundation. One of the solutions of this problem is participatory sensing [58][59]. By adopting the participatory sensing, the city does not need to install the many sensors to all around the city, because the citizens help to offer not only the information that conventional sensors can sensing but also dynamic events such as traffic accidents or illegal dumps. In this framework, citizens who participate to offer the information can be regarded as sensors, which is so-called human as a sensor.

As introducing participatory sensing, though, not only the economic cost remains but also the new cost is incurred: human cost. Citizens would participate to offer the information at the beginning of the participatory sensing system, thanks to their curiosity and interests. However, with losing the interest and curiosity, it is difficult to request citizens to continue to offer the information. There is a lot of works to tackle this problem in the terms of incentive [60], which motivates the citizens to participate in the information offering continuously. Moreover, one of the problems mentioned among the studies of participatory sensing is privacy concerns. When some user report issues about the city with their location information, the other users can infer or know where the user was. Repeating this inference, again and again, the range of the user behavior is easily estimated. This is the case to avoid and users become hesitate to report issues frequently.

Thus, in this research, we propose an alternative approach not to request users to report urban information with their location but to obtain the location from other sources, that is, social network service. The reason why we use social network service is simple that users already have offered information about the city willingly. Zheng et al. [4] also have claimed to utilize social network service as passive crowd sensing; human as a sensor. However, they have only focused on the social network service which enables users to attach location information, so-called location-based social networks, which is much fewer than general social network service; it is only 5 % of all tweets that attached in Twitter[2]. In order to remedy this trouble, we have to infer the location of users by analyzing the text which users posted to social network service, and regarded as 'pseudo' human as a sensor.

Urban Information Extraction

Given a human as a sensor, while it is convenient to capture the dynamic phenomenon which happens unscheduled so that it is difficult for conventional sensors to capture it, there is a problem that citizens cannot capture them continually in spatial-temporal. The one of the way that complements the lack of humanity as a sensor is *facility as a sensor*, which utilizing the facilities that have already been installed in the city, such as surveillance camera or the sensors which automobile equips. If we can utilize those facilities exists in the city efficiently with conventional sensors and human as a sensor, we can cover the whole area of the city to sense the status of the city. Furthermore,

[2] `https://www.twitter.com/`

other devices, so-called IoT devices or edge devices, would be developed in the future so that enormous devices would exist in the city.

While utilizing these devices are well-suit for affordable urban computing, there is a difficulty in terms of data processing where those devices are. In general, such those devices connected with each other in an application, the devices are synchronized as to time. Therefore, collecting and saving the data to the storage such as a database is not difficult; indexing the data with its generated timestamp. Meanwhile, the edge devices such cameras are not synchronized so that treating the data generated from the devices is troublesome. In order to handle this situation, conducting the process in the devices as much as possible is effective. Indeed, affordable urban computing requires edge devices to conduct some processing within itself.

The next problem is that we need to finish conducting some processing completely within the edge devices in order to make the data management easier. We assume that the processing is image processing such as using deep convolutional neural networks. This is because if the degree of sensing abstraction of data for urban computing becomes deeper than that of raw data that is currently dismissed, such high abstracted data is useful. For instance, though images from a dashboard camera are now dismissed due to the Personal Information Protection Law, the road damage data detected from the images is valuable. However, most of the studies using deep conventional neural networks assume that they work on a high-performance computer which equips GPU. In short, this is the question that contains 1) can the deep neural networks extract the high-abstracted information from data? and 2) does such deep neural network enable to work on edge devices?

3.4.2 Development and Management Problem

In the description of the edge device as a sensor, we refer to the problem of edge device environments that it is difficult to manage the data generated from edge devices. Even if the processing can be conducted within the devices, we need to manage the result of that processing. Simultaneously, developing and deploying the processing/applications to the enormous amount of edge devices has a lot of trouble with coding; treating their network environments or their specification.

Therefore, we need to provide a development environment that enables users such as not only the developer but also city officers to develop and manage the data flow and devices with arbitrarily contain. Moreover, since we assume that our affordable

urban computing can be applied to any cities, the source developed by someone in this development environment should be reusable and it should be easy to use.

Programming languages and toolchains that have evolved to facilitate the development of traditional applications, in sometimes, lack the features needed to address, and in some cases exploit, the scale and dynamic nature of affordable urban sensing. Especially, we consider that the components that represent the functions should be used easily, the communication among the components should be abstracted, and it should be suitable for a range of skills, such as machine learning approach.

3.5 Summary

Conventional urban computing has the potential to solve various city challenges, yet some city challenges are still remaining as being unsolved due to coarse-grained sensing. However, it is difficult to solve the challenges by defraying more costs, so that we need to consider alternative urban computing to solve them. In this chapter, we proposed the affordable urban computing which enables us to monitor and understand the status of the city with fine-grained sensing approach. In order to realize this new urban computing, we described three problems: modality translation problem, edge execution problem and development, and management problem. Following chapters describes the solution of these problems, respectively.

Chapter 4

GeospaceMapping: Urban Transfer Sensing

4.1 Introduction

This chapter presents urban transfer sensing technogies that estimates the location of urban data, especially texts and images generated by human, to transfer the non-geospace data into geospace data. With this technology, we can obtain the data related to geospatial-temporal aspects with reference to urban computing. In this thesis, we suppose the dynamic urban data contains the event that it is difficult for us to predict that event happens or the detail of the event, such as how many people are in the event or where people come from. Understanding the dynamic urban context is very important for the citizens and the administrative organ to tackle the issues that cities face, e.g. natural disasters, traffic congestion. Although such dynamic urban context is very useful to understand the status and teh context of the city, most of the data does not contain the geo-spatial information, such as geo-coordinates: latitude or longitude.

One of the approach to collect the information which indicates geospace and the content of the city is the participatory sensing [59] that is the concept of gathering the data not from sensors but from users. It is difficult to capture the dynamic event by sensors due to the characters of events. On the other hand, since users can be adapted to the dynamic event, they can report the events as the data. Although the participatory sensing seems to be suitable for capturing the dynamic events, there are two big problems to be solved: the incentive and the privacy concerns. As the early stage of some participatory sensing application, many users would volunteer to provide the data about the dynamic events thanks to their curiosity. After the early stage, the incentive is required in order to offer the users to continue to provide the data. Simultaneously, providing the data about the event related to the region such as local festival would violate the privacy concerns; we can know that the users are there. In short, the data from the participatory sensing application would be selectively biased, since the users very carefully select the opportunity of providing the data.

As the platform that the data is provided by users as same as the participatory sensing, we can use the data from social network services (SNS). Since the users in the SNS tend to post text and/or images that describe their experiences, we can know what happens around the users. By contrast, the geolocation informations are not posted to the SNS as well as the text and images due to the privacy concerns. For example, only 1% of tweets contains the geolocation on Twitter. If we can know the geolocation of the information that users post in the SNS, it helps us to understand the urban context.

To tackle the geolocation missing, we propose the method to estimate the location

(a) Tokyo Disney Resorts. An attraction is crowded now.

(b) Apple Store in Ginza. Long lines form on the day that iphone 5 go on sale.

Figure 4.1: Examples of geotagged tweets. We can know what happen and where it is through observing geotagged tweets.

of tweets where they are posted. The remains of this chapter describe the method to estimate the location of users who posted tweets in detail firstly. Utilizing the method, we can gather the information about the event, the location and the text describing the context. Then, we introduce the two kinds of application that can be developed by acquiring the geolocation information. The first application is that investigating the popularity of events that what kinds of a genre that the participants of the events have. With this application, we can apply to the promotion or advertisement for events.

The second application is that estimating the transportation modes of users what kind of vehicles (or walking) is used to move around in the city by users. With this application, the city governments and companies can know the status of public authorities and plan to replace the new timetable for buses or trains. In each section that describes the two application, we formulate the applications as machine learning tasks; supervised classification tasks. After we explain the approach to solve the each task, we describe the dataset which we used in the experiment.

Finally, we discuss the position of our GeospaceMapping in the context of urban computing.

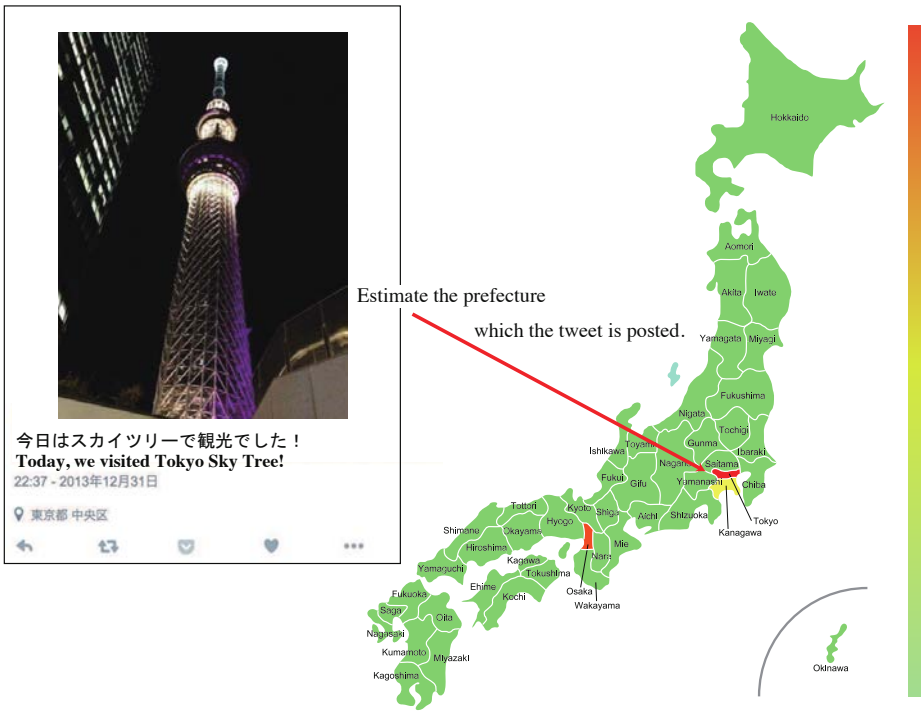


Figure 4.2: Overview of location estimation. Given the text of geotagged tweet, we can estimate which administrative district the tweet is posted.

4.2 Location Estimation by Textual Information and Visual Information

To understand what happens in urban space, one useful data come from tweets that are posted on microblogs called Twitter[1]that imply rich information about a city. Thanks to the spread of information technologies, such as social networking service (SNS) and intelligent mobile devices like smartphones, a lot of users post tweets as well as report experiences in their urban lives. As describe in the previous seection, some users post tweets as well as report experiences in their urban lives with their location detected from GPS, called geotagged tweets, so that we can understand what and where they experience. In particular, some users post tweets with their location detected from GPS, called geotagged tweets, so that we can understand what and where they experience. Figure 4.1 shows examples of geotagged tweets. We can know the attraction *"Center*

[1]http://www.twitter.com

of The Earth" in Tokyo Disney Resorts was crowded when the geotagged tweet was posted in Figure 4.1(a). And also in Figure 4.1(b), we can know the long lines formed in front of Apple Store in Ginza on the day that iphone 5 went go on sale. Although the geotagged tweets are very valuable and informational, most users tend not to post them because of privacy concerns.

To address this issue, many studies have estimated the location of non-geotagged tweets so that those can be treated and used as geotagged tweets. Basically, most of the studies have analyzed the text of non-geotagged tweets so that we can estimate where the non-geotagged tweets are posted as shown in Figure 4.2. One of the traditional methods for location estimation is a corpus-based approach that creates the corpus mapping between words and location in advance. Although it is very intuitive and easy to apply, the corpus-based approach requires updating the corpus as a new word appears; thus the cost of maintenance becomes high. Due to the variety of words being used at tweets such as for Internet slangs, the cost of corpus maintenance is much higher than articles such as newspapers.

Another traditional way of addressing the issue is a machine learning approach. In this approach, the *bag-of-words representation* (BoW) is often used for inputs, which is a technique in natural language processing (NLP) and represents a text as a set of its multiple constiuent words [61]. The BoW into which the tweets are translated is used as an input to machine learning method, e.g. *support vector machines* (SVM). Since the number of characters in a tweet is restricted to 140, its information content is not sufficient for input. To increase the information content, many studies have leveraged other resources [62, 63, 64]. Although they have estimated the location of tweets, these approaches require designing features for inputs and prior knowledge such as NLP. Although the traditional methods for location estimation, such as corpus-based and machine learning, have been introduced, applying these methods to geotagged tweets require updating the corpus as a new word appears, designing feature for inputs to machine learning, or leveraging other resources [62, 63, 64, 61]. we depict the actual geo-tagged tweets which were collected in our dataset in Figure 2. There are four kinds of tweets that are classified with a combination of two indicators. One indicator is whether the text of tweets contains the words that indicate somewhere such as place or facility name. Another indicator is whether some landmarks are in images, for example the tower. The first type of tweets is shown in Figure 2(a). We can find this tweet is posted in Tokyo prefecture, because in this tweet, there are key word “ Tokyo

Tower ” that represents the landmark of Tokyo prefecture, and also it is in the image This tweet is good example that it is easy for either people or machine to estimate the location. Simultaneously, estimating the location of the tweet depicted in Figure 2(b) is also easy as well as (a), for the tweet contains the key word that indicate the location such as “ Haneda Airport ” that is famous among Japanese. In summary, the common point between Figure 2(a) and (b) is that both kinds of tweets contain some words representing some place explicitly. Although the location of these tweets can be estimated by previous work, dictionary-based method, we propose the method using the extension of deep neural networks, recurrent neural networks. Our model estimate the location without efforts that previous work has exerted because they learn which words are important for location estimation.

In contrast to Figure 2(a)(b), there is nothing that indicate somewhere directly in the tweet depicted in Figure 2(c) and (d). However, we can infer that the tweet depicted in (c) was posted in Osaka prefecture because there are hints in tweets and if one knows there is a Ferris wheel in Osaka. It is significant to know the objects which are depicted in. Note that it is difficult to say the user whether they intend to take a picture with those objects or not (it is obvious that our example in (c) does so). At the example in (d), it is almost impossible to estimate the location, because there is no hint in text and image with just represented for dinner. While it is difficult for people to know all objects in the world, machine can know them with a lot of data. Thus, studying how machine can estimate the location of those tweets such as depicted in Figure 2(c) is very important. We propose the model that is able to treat not only texts but also images, additionally. We explore a model which learns effective mixing of texts and images for better location estimation.

As described in this section, in order to estimate the location of tweets without any prior knowledge and feature engineering, we apply bidirectional long-short term memory (BLSTM) [65], which is one of the *recurrent neural networks* (RNNs), for location estimation task. The task is considered as a multi-class classification problem where geotagged tweets are classified into administrative districts. In our experiments, we collected geotagged tweets that are posted in Japan, and classified them into administrative districts. As a result, our approach could classify with higher accuracy than baseline methods. Simultaneously, the BLSTM network could learn the conceptual knowledge about administrative districts, for instance, the name of specialty.

Table 4.1: The dataset of geotagged tweets.

Month	Tweets	Users	Users Freq.		After trimming		Image attached	Prefecture included?	
			≤10	500≤	Users	Tweets		YES	NO
2013/11	22938345	225845	179558	121	46166	17426655	190445	25198	165247
2013/12	52670853	293624	206867	633	86124	41977134	495595	58128	437467

4.3 Approach

One of the easiest methods to estimate the location of tweets posted by users is to find the name of place or spots in the text of a tweet and to match it with geographical corpus. However, this approach is difficult to be applied on abbreviated expressions for specific places. Therefore, numerous attempts with machine learning technique have been undertaken. Roller et al. treated this location estimation task as a multi-class classification task, which is explored the same as in this study, and classified the geotagged tweets by using bag-of-words (BoW) [61]. For the properties of tweets, the frequency of appearance for each word in a tweet is very low but diverse and the vector in BoW that represents tweets becomes very sparse, which makes the location estimation task more difficult. To address this problem, there are a lot of studies to increase the information of tweets [63, 62, 64]. However, these approaches required to design the input features and expert knowledge so as to estimate location successfully.

One of the deep neural network (DNN) architecture, on the other hand, RNNs have been adopted to a variety of NLP task because of their high performance [66, 67]. Moreover, deep convolutional neural networks have achieved state-of-the-art results for analyzing sentiment of short text messages, found such as in Twitter, as positive or negative [68, 69].

In summary, most previous approaches to estimate the location of tweets has employed a large amount of feature engineering and additional resources. Meanwhile, the use of DNNs has succeeded in numerous NLP tasks without the aforementioned efforts. Prior work has shown that some DNNs are valid for tweets and RNNs are well-suited for NLP tasks[66, 67], however, to the best of our knowledge ours is the first method applied to tweets that directly takes a classification approach to location estimation using RNNs.

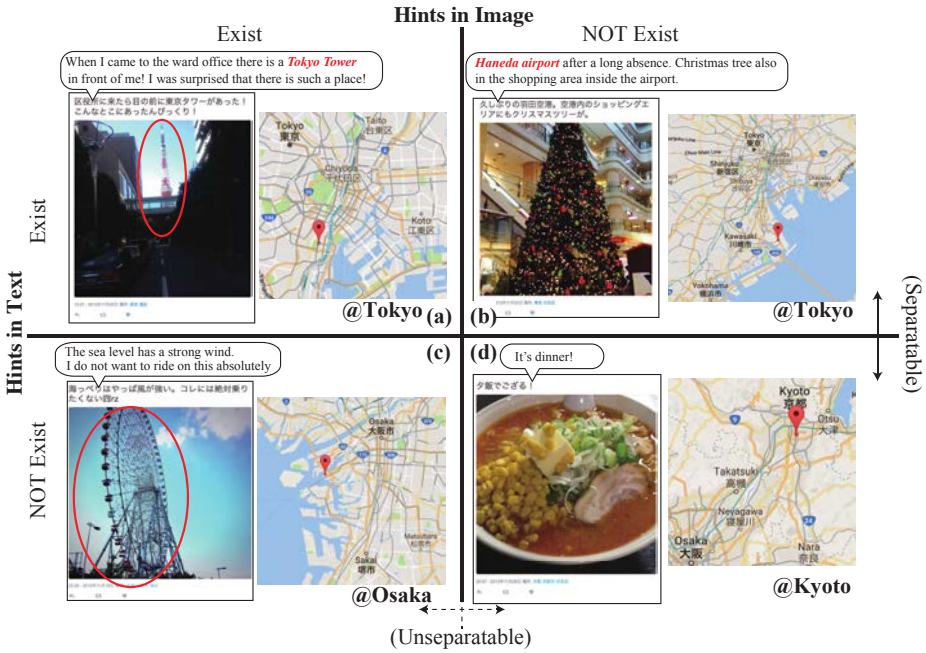


Figure 4.3: The geo-tagged tweets we collected during November, 2013. We can classify these tweets to four kinds by their containing information, concretely, such as the hints about location in texts or images. (a)this type of tweets has both hints, for this example, there is “Tokyo Tower” in the text and in the image, so that human can estimate the location of this tweet would be Tokyo prefecture. (b)The tweets that contains the hints in the text and not in the images. Although this tweets lack the hints from images, human can still estimate the location because of the text, for instance “Haneda airport”. (c) these tweets do not contain the hints in the text but in the images. It might be more difficult to estimate the location than the tweets of (a), (b), but still can. Lastly, (d) is almost impossible to estimate the location, since these tweets do not have any hint in their media.

4.3.1 Long-Short Term Memory Units

LSTM units [50] as shown in Figure 4.4 have been employed to produce promising results on a variety of NLP tasks. Since the tweets are often written in unstructured text, a model is required that accumulates word information which is an important factor in location estimation and discards which is not. Therefore, LSTM architectures seem like a good fit to tweets. The input is the texts of a geotagged tweet, which is a sequence of N words (encoded in a 1-of-k representation), and each word is transformed into an arbitrary-dimensional vector x_t with the index $t = 1 \ldots N$ to denote the position of a

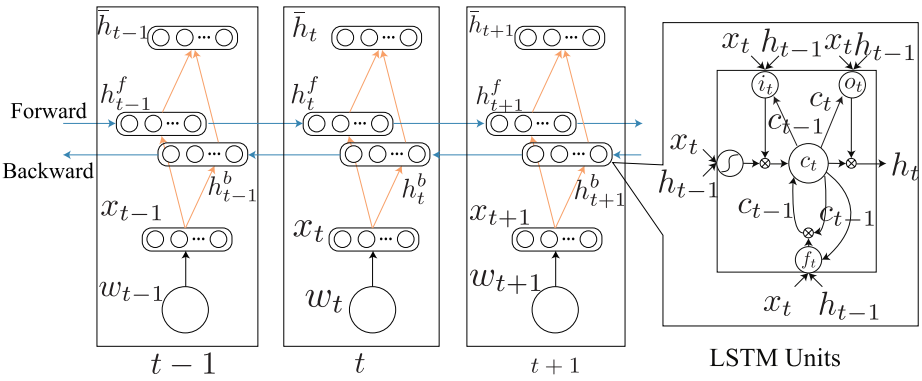


Figure 4.4: A bidirectional LSTM network.

word in a tweet. They are implemented as the follows:

$$i_t = \sigma(W^{ix}x_t + W^{ih}h_{t-1} + W^{ic}c_{t-1} + b^i) \tag{4.1}$$

$$f_t = \sigma(W^{fx}x_t + W^{fh}h_{t-1} + W^{fc}c_{t-1} + b^f) \tag{4.2}$$

$$o_t = \sigma(W^{ox}x_t + W^{oh}h_{t-1} + W^{oc}c_t + b^o) \tag{4.3}$$

$$c_t = f_i \odot c_{t-1} + i_t \odot \sigma(W^{hx}x_t + W^{ch}h_{t-1} + b^h) \tag{4.4}$$

$$h_t = o_t \odot \sigma(c_t) \tag{4.5}$$

where σ is the sigmoid function, $\odot$ indicates the element-wise product, and i, f, o and c are the input gate, forget gate, output gate and cell vectors respectively. The weight matrix W and the respective biases b subscripts have the meaning as the name suggests.

4.3.2 Bidirectional LSTM Networks

BLSTM (Figure 4.4) is the model which is constructed by combining LSTM which was described in the previous section, with bidirectional recurrent neural networks (BRNNs) [70]. The difference between BRNNs and RNNs is that BRNNs consist of two independent streams of processing, one pass forward (h^{ft}) and the other backward (h^{bt}), while RNNs pass only forward. Therefore, the formula of hidden layers is adjusted:

$$h_t^f = \sigma(W^{h^f x}x_t + W^{h^f h^f}h_{t-1}^f + b^{h^f}), \tag{4.6}$$

$$h_t^b = \sigma(W^{h^f x}x_t + W^{h^b h^b}h_{t+1}^b + b^{h^b}), \tag{4.7}$$

$$\overline{h}_t = \sigma(W^{hh}(h_t^f + h_t^b) + b^y). \tag{4.8}$$

4.3.3 Image Encoder

We propose the model which inputs are images, called image encoder(IENC). Image encoder is inspired by convolutional neural networks (CNNs), which is a special type of a feedforward neural network, or multilayer perception, and has been designed to work well with two-dimensional images. Image encoder consists of multiple convolutional layers followed by a few fully-connected layers as shown in Figure 4.5. At each convolutional layer, the input image of width n_i, height n_j and c color channels (usually RGB channels) represented as $\mathbf{x} \in \mathbb{R}^{\mathbf{n_i} \times \mathbf{n_j} \times \mathbf{c}}$ is first convolved with a set of local filters $\mathbf{f} \in \mathbb{R}^{\mathbf{n'_i} \times \mathbf{n'_j} \times \mathbf{c} \times \mathbf{d}}$ whose width is n'_i, height is n'_j and d kinds. For each pixel (i, j) of $\mathbf{x}$, we get:

$$\mathbf{z_{i,j}} = \sum_{\mathbf{i'=1}}^{\mathbf{n'_i}} \sum_{\mathbf{j'=1}}^{\mathbf{n'_j}} \mathbf{f}(\mathbf{f}_{\mathbf{i',j'}}^{\top} \mathbf{x_{i+i',j+j'}}), \tag{4.9}$$

where $\mathbf{f_{i',j'}} \in \mathbb{R}^{\mathbf{c} \times \mathbf{d}}$, $\mathbf{x_{i+i',j+j'}} \in \mathbb{R}^{\mathbf{c}}$ and $\mathbf{z_{i,j}} \in \mathbb{R}^{\mathbf{d}}$. f is an element-wise nonlinear activation function. The convolution in Eq. 4.9 is followed by local max-pooling:

$$h_{i,j} = \max_{i' \in \{ri, \ldots, (r+1)i-1\}, j' \in \{rj, \ldots, (r+1)j-1\}} \mathbf{z_{i',j'}} \tag{4.10}$$

for all $i \in \{1, \ldots, n_i/r\}$ and $j \in \{1, \ldots, n_j/r\}$. r is the size of the neighborhood pixels. This max-pooling operation has two properties. First, it reduces the dimensionality of a high-dimensional (high-resolution) output of the convolutional layer. Furthermore, this operation summarized the activation of the neighborhood feature activations. After twelve of convolutional layers, the final feature map from the last convolutional layer is flattened to form a vector representation $\mathbf{h}$ of the input image. Then, this vector $\mathbf{h}$ is fed through two fully-connected layers. The architecture of image encoder is depicted in Figure 4.5. We apply batch normalization to all layers in image encoder.

Recently the CNNs have been found to effective at the image recognition tasks. For instance, the ImageNet Large Scale Visual Recognition Challenge has a classification track where more than a million annotated images with 1,000 classes are provided as a training dataset. In this challenge, the CNN-based method outcomes other non-CNN-based method [46, 71, 72, 73]. Therefore, we expect our image encoder to perform well in our study.

4.3.4 Multimedia Encoder

Finally, we propose the multimedia encoder, which consists of text and image encoder and is able to get both texts and images.

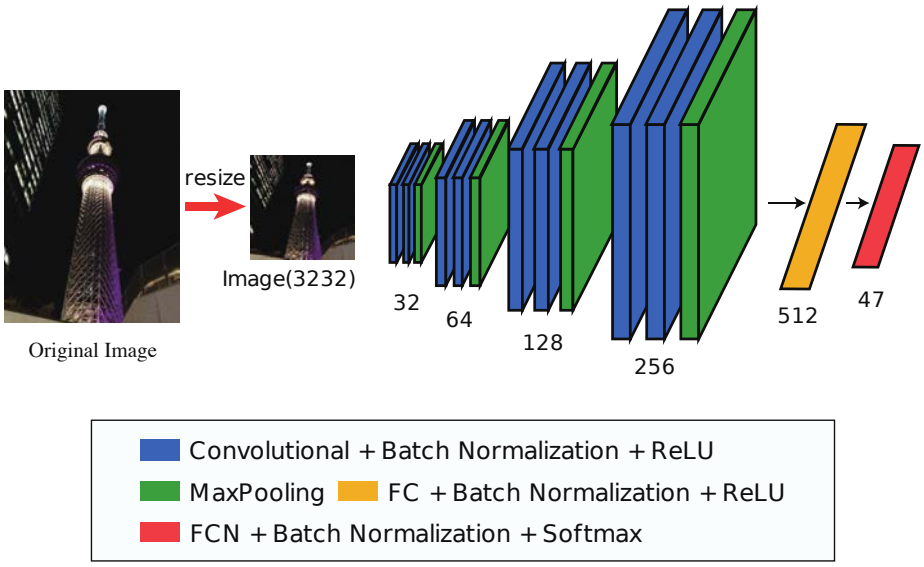


Figure 4.5: Image encoder architecture. The image is resized to 32×32, and then the resized image is taken as input. our image encoder consists of twelve convolutional blocks and two fully-connected networks. In each convolutional block, the activation function of convolutional is rectified linear function (ReLU) and the image is convolved twice followed max-pooling operation. Batch normalization is applied to all layers.

Once text encoder and image encoder are trained on texts and images respectively, we can use any intermediate representation, such as the feature map from any recurrent unit, any convolutional layer or the vector representation from any subsequent fully-connected layers, of the whole networks. This concept can be considered as transfer learning, which is observed that the use of these intermediate representation from the neural networks as a data descriptor significantly boosts various recognition tasks [74, 75]. We design two types of architecture of multimedia encoder, multimedia mixture encoder and text-based multimedia encoder.

Multimedia mixture encoder (MMENC).

In this model, we attempt to mix the features from each input. To do this, after we get the output vector from each encoder that summarize each input, we concatenate both vectors to create one vector. Then we input this vector to a few fully-connected networks. The formula of this model is as followed:

$$h_i = FC(concatenate([LSTM(\text{text}_i); CNN(\text{image}_i)])) \tag{4.11}$$

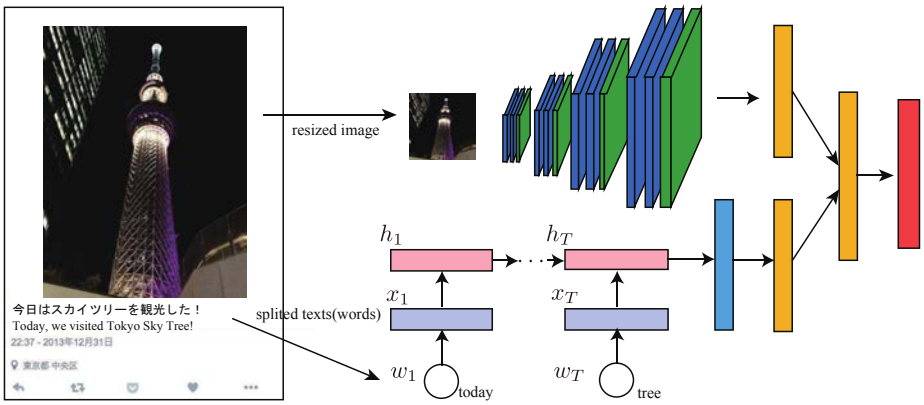


Figure 4.6: The architecture of multimedia mixture encoder. This model is designed to mix the features of each input, texts and images. Text encoder and image encoder encode the input to vectors, and then concatenate these vectors. The concatenated vector is taken as input to fully-connected networks and output the location estimation result.

$$y_i = \psi(Wh_i + b), \tag{4.12}$$

and the architecture of this model is shown in Figure 4.6.

Text-based mixture encoder(TMENC).

Also, we consider the images might support the words to represent the location. In order to present this notion, we concatenate the vector which represent the image with to each word vector. Therefore, we change the formula in Eq. 4.1 to:

$$h_t = \sigma(W^{hx} concatenate(x_t; CNN(image_i)) + W^{hh} h_{t-1} + b^h). \tag{4.13}$$

The architecture of this model is shown in Figure 4.7.

The weights of these multimedia encoder are initialized with that of text and image encoder after trained. This idea comes from pre-training [76, 77, 78].

4.3.5 Model Training

In order to solve location estimation task as a multi-class classification task, we assign a label for each tweet to indicate the administrative districts where it was posted. To predict the class distributions, we set the model output as $y_t = softmax(W^{yh} h_t)$. The softmax layer's outputs are interpreted as conditional probabilities $y_{nk} = p(w_{\leq T_n})$;

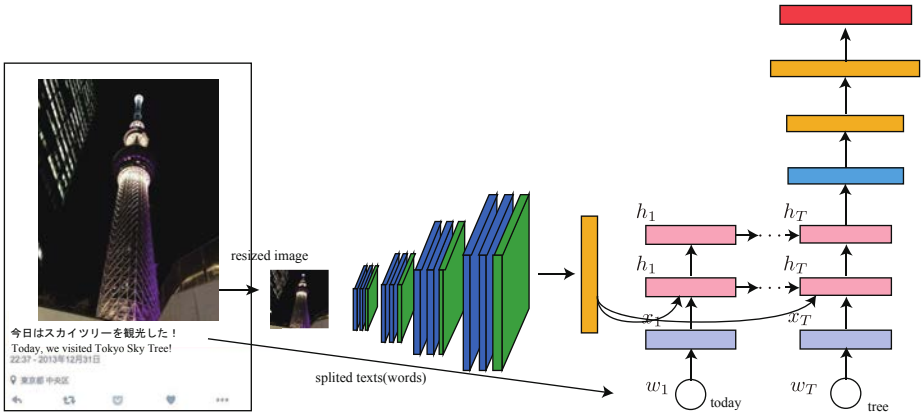


Figure 4.7: The architecture of text-based mixture encoder. This model is based on text encoder except the image is also taken as input. Before input to LSTM unit, the image vector from image encoder is concatenated with word vector.

hence the objective which the model is trained to minimize becomes cross-entropy. The parameters of the model are optimized by Adam [79].

4.4 Experiment

In this section, we describe the experimental setup and results. For our experiment, we used GPU NVIDIA GeForce 980 and implemented the model by theano[2] library.

4.4.1 Dataset

The dataset we used is the geotagged tweets posted in Japan from November 1st, 2013 to the December 31th, 2013. First, we removed the users who posted geotagged tweets less than 10 times, and then more than 500 times to exclude the bot accounts. Finally we elicited the geotagged tweets to which images were attached, assuming that users attempted to inform with not only text but also images.

If the tweets contain the name of prefectures, it would be easy for the model to correctly classify the location of the tweets into prefectures. To prevent making the tasks easy, we separated the geotagged tweets that contained the name of the prefecture from those NOT. The result of the dataset after scraping is shown in Table 4.1. We excluded the emoticon and alphanumeric character and applied morphological analysis

[2]http://deeplearning.net/software/theano/

Table 4.2: Experiment environment

OS	Windows 8.1
CPU	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.4GHz
Memory(RAM)	32G
GPU	GeForce GTX 980
GPU Memory	20G
Language	Python v2.7
Library	Theano, Keras

Table 4.3: Examples of tweets excluding prefecture name which was estimated correctly.

Case	Text of tweets	(GT) Pref.
1	Camping out with the sound of the sea. Tomorrow I'll go to *Mt.Kaimondake*.	Kagoshima
	The sea trip by a *Matsushima* sightseeing boat, I took a picture of colleague!!!!! Black-tailed gull catch the raincoat shrimp thousant	Miyagi
2	My wife is shopping now(^^;; @ touch *Gotemba Premium Outlets* with a picture!	Shizuoka
	Country Farm Tokyo German Village♡	Chiba
3	Hey, everyone, let's go to *Okikoku Festival*!(≥ ∧ ≤)	Okinawa
	I have eaten *Hitsumabushi* cooked by Charcoal!! It was soooooo delicious♡	Aichi
	Lunch time with *Genghis Khan*	Hokkaido

to extract the nouns from each geotagged tweet.

Before classifying geotagged tweets, we transform geotags to the corresponding prefecture labels by Geohash[3]. We used the geotag of prefectures from Japanese location reference information[4] provided by Ministry of Land, Infrastructure, Transport and Tourism and represented prefecture as 1-of-k 47-dimensional vector.

4.4.2 Result of location estimation

In this experiment, we used two types of evaluation indicators, the classification accuracy and the error distance, because the required information depends on the application. The classification accuracy $Acc@N$ shows the performance of how the model can classify the geotagged tweet into prefectures correctly in top N, and the error distance indicates the mean and median distance calculated between the geotag of the tweet and the center point of the prefecture.

In this experiment, we considered four kinds of baseline methods. The first method is random (RND), which is classifying the tweets into the prefectures randomly. The

[3]http://geohash.org/
[4]http://nlftp.mlit.go.jp/isj/index.html

Table 4.4: Examples of tweets excluding prefecture name which was estimated incorrectly.

Text of tweets	Answer Pref.	Estimated Pref.
I'm making scone now(-▽-)	Hyogo	Ibaraki
I'll go to Taiwan tomorrow! Does anyone have something to say for me∗(ˆoˆ)/∗!?	Saitama	Nara

Table 4.5: Location estimation accuracy (%) with the tweets including prefecture name.

Method	Accuracy (%)			Error Distance(km)	
	Acc@1	Acc@2	Acc@3	Mean	Median
RND	2.152	4.365	5.878	516.68	434.10
MFA	7.862	14.509	20.486	401.46	343.03
PMC	72.783	-	-	504.30	419.81
SVM	**90.699**	-	-	61.555	25.564
LSTM	89.434	91.939	93.229	64.125	25.452
BLSTM	90.203	**92.733**	**93.676**	**61.453**	**24.988**

second method is most frequent appearance (MFA) that classifying all tweets into a prefecture which appears most frequently in training data. The third method is pattern-match based on the corpus (PMC) where the tweets are classified according to the name of prefectures mentioned. We applied this method only to the dataset that includes tweets with the name of prefectures. And the last method is SVM where the BoW of tweets are used as input.

Table 4.5 presents our result with the geotagged tweets that include the prefecture names and Table 4.6 with that of not including the name of prefectures. Our model, BLSTM outperformed all baseline methods except SVM in the tweets including the prefecture. This is remarkble, given the simplicity of our approach and the lack of feature engineering. Comparing between TENC and IENC, we can say that texts are more useful than images to estimate the location. It is reasonable because it is easy for user to indicate the location by language rather than pictures. Comparing Multimedia encoder (MMENC and TMENC) with TENC, adding images information is effective although estimation accuracy with only images (IENC) is worse than with only text (TENC). Lastly, comparing between MMENC and TMENC, TMENC is better than MMENC. We think that the using image information as the additional source for text information is better rather than treating both information as equivalent.

Meanwhile, we discuss about location estimation with texts and images. We show

Table 4.6: Accuracy obtained on the test set for our location estimation task with geo-tagged tweets.

Method	Acc@1	Acc@2	Acc@3
RND	2.152	4.043	6.237
MFA	17.12	28.08	34.44
SVM	23.40	—	—
TENC	35.34	40.72	46.43
IENC	33.32	38.32	42.94
MMENC	36.52	42.48	49.82
TMENC	**37.72**	**44.13**	**50.21**

the examples for the correct and incorrect cases in Figure 4.9. The example shown in Figure 4.9(a) is the type of Figure 4.3(a) that the hints about the location are in both texts and images. Simultaneously, Figure 4.9(b) shows the type of Figure 4.3(c) that the hints are only in images and not in texts. The ground truth of these two examples is Chiba prefecture, where is Mickey Mouse and Cinderella Castle, Tokyo Disney Resort, and our model estimate the location correctly. Although at the example in Figure 4.9(c), our model estimated the location of it as Chiba prefecture naturally, the ground truth is not Chiba prefecture but Niigata prefecture. We consider that the user of this example posted it not at Tokyo Disney Resort but when he/she went back to home, he/she did. We cannot understand why he/she posted tweets with the geo-tag of the other place, but it may be considerable that our model cannot estimate the location correctly. Note that there are a lot of pictures of "selfy" or foods (such as Figure 4.3(d)) and it is also impossible to estimate from images. In future work, the part of multimedia encoder to treat image, IENC, can be replace other CNN architecture [80, 81], so that model can use some texts is in the images, for instance, the shop name or street name.

4.4.3 Evaluation of during the model training

In this experiment, we evaluated how the accuracy changed when the percentage of the dataset whose tweets include the name of prefectures and the size of training data increased. To evaluate it, we fixed the size of training data as 20157 and increased the percentage of tweets which include the name of prefectures. The result is illustrated in Figure 4.8. We can observe that when the model trained the same size of tweets both including the name of prefectures and not, the accuracy becomes the highest of all. Figure 4.10 shows the training error, validation error and test accuracy as the size

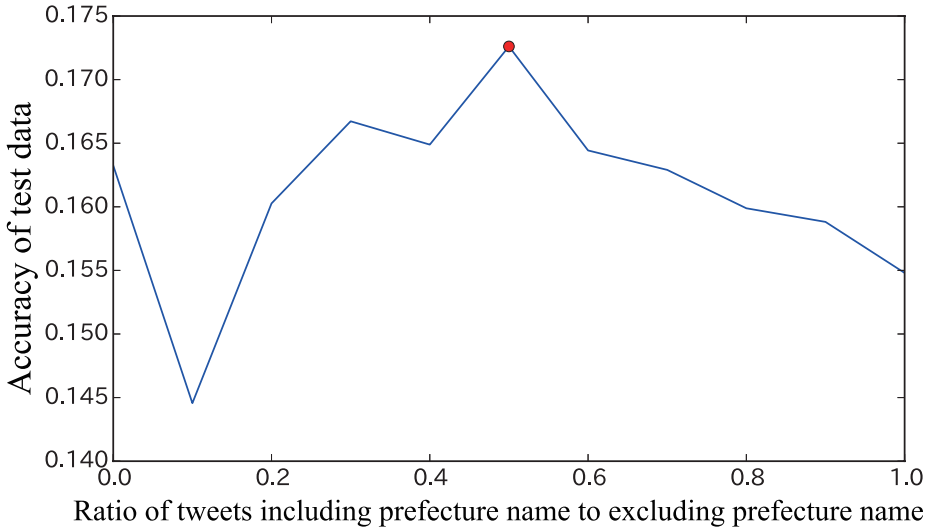


Figure 4.8: The estimation accuracy change. When the amount of tweets containing Pref. name and that of not is same, the accuracy becomes highest.

of training data increases. Our results show that increasing the size of training data, impacts the accuracy of location estimation to be high (Figure 4.10 green line), while the runtime increase is $O(n)$ in Figure 4.11.

4.5 Application 1: Event Popularity

In this section, we present an application that using location-based social networks to understand about events occuring in the city. Although there are many previous works that attempted to discover and classify urban events based by analysing SNS, urban events classification by their name or scale is not sufficient when these works are applied to recommendation system. To provide more aspects of urban events to users, it is necessary to classify urban events with more indicators. Another problem is calculation amount for the analysis. To achieve real-time recommendation, the analysis should be light-weight and quick. Furthermore, the SNS such as Twitter has several API limitations. For example, we can acquire the information of users only 15 persons per 15 minutes[5]. Therefore, it is important to provide urban events classification method based on light-weight calculation and light-weight access to SNS. To solve

[5]https://dev.twitter.com/twitter.com/docs/api/1.1/get/friendships/show

(a) Text: ○, Image: ○→Chiba
Correct

(b) Text: ×, Image: ○→Chiba
Correct

(c) Text: ×, Image: ○→Niigata
Fail

Figure 4.9: The examples for the correct case (a)(b) and incorrect case (c). Although in these three cases every tweet represents Tokyo Disney Resort and our model estimated the location of them as Chiba prefecture, (a) and (b) case is correct but (c) case is not because of their label being Niigata. ○ denotes there are some hints about location and × denotes there is no hints about location.

these problems, our research goal is to present a new indicator for urban events classification, and also it's analysis method with minimised calculations and API access to SNS. By achieving the research goal, we focus to create various applications such as event recommendation or city management.

4.5.1 Popularity of Urban Event

We propose a new indicator for classification of urban events, called *Popularity*. Generally, *Popularity* means the state or condition of being liked, admired, or supported by many and various people. As an example of high popularity events, at fireworks festivals, the range of participants' generation must be very wide and their hobbies or interests are also diverse. Conversely, urban event such as specific artists live concert (e.g. heavy metal) would has low *Popularity*, since the participants might be specific fan of the artists or genre. Thus, in our work, we defined *Popularity* as the diversity of participants in urban events. If we can estimate *Popularity* in each urban event, the recommendation system is able to know whether the target user are suitable. When the urban event has high popularity, the system would recommend various users. But if not, the system would select the characteristic users to recommend.

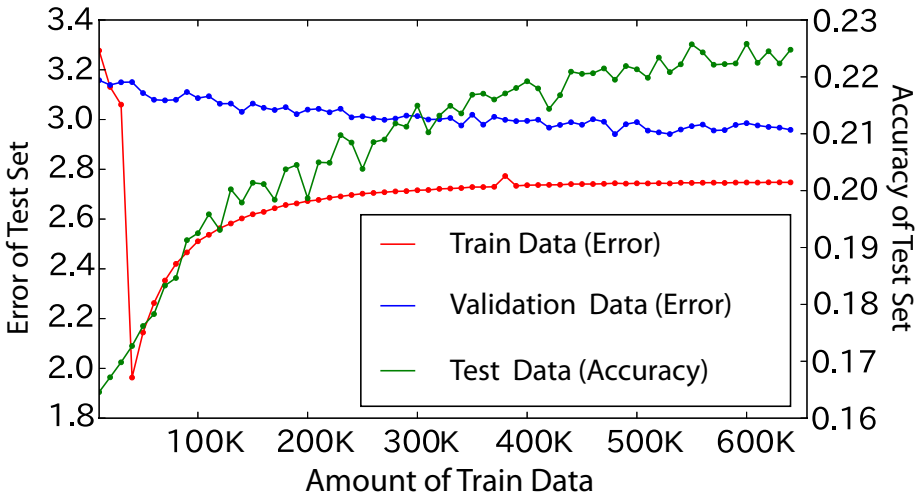


Figure 4.10: Training error, validation error and test accuracy. The more the size of training data increases, the higer test accuracy rise.

To estimate the Popularity of the event, it is necessary to profile the attributes of participants. In Twitter, it is possible to fill the profile in each account, so the system can get the attributes of the participants by analysing profile. But the users who fill the profile are not so large because filling the profile leads the privacy concerns or is just bothered for users. Therefore it is difficult to understand participants' attribute by analysing Twitter accounts' profile. In our study, we focus on *Friends* of participants. *Friends* of participants are defined as the account which participants follow in Twitter. In general, users follow the accounts of the friend in a real world or of a famous user who posts the information which users are interested in. Since we assume that *Friends* of the users reflect the attributes of users, we analyse *Friends* information of urban events' participants to understand the participants' diversity. Our assumption of the relationship between *Friends* and *Popularity* is shown in Figure 4.12. In urban event being low *Popularity*, such as the characteristic community party, the participants would have similar attributes, and they would follows mutual *Friends* in Figure 4.12(a). On the contrary, in urban event being high *Popularity*, such as fireworks festivals, the attributes of participants would become diverse, and the following *Friends* would be disperse in Figure 4.12(b).

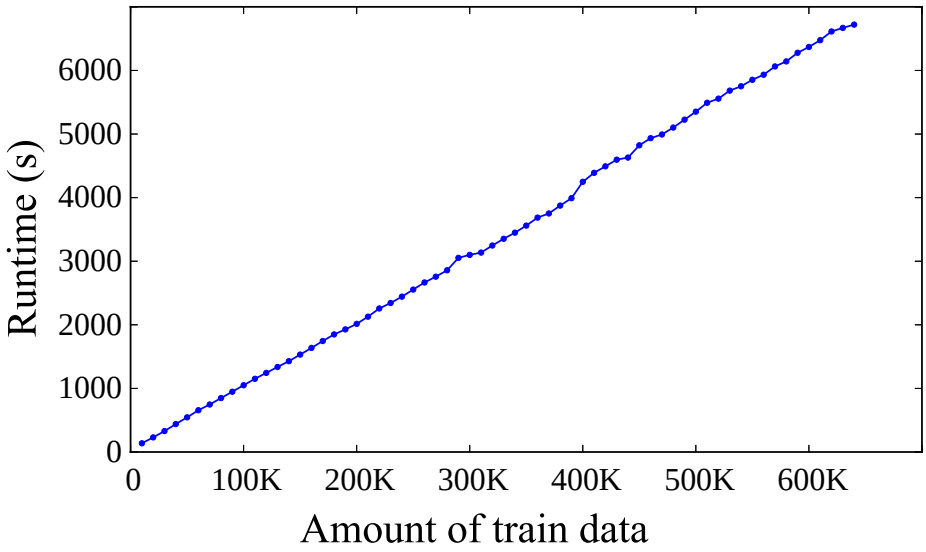


Figure 4.11: Runtime during the model training. The time rise in $O(n)$.

4.5.2 Analysing Method

We represent the analysis procedure based on assumption. 1)First we collect the all of *Friends* f_{p_k} of the participant p_k who belongs to the set of all event participants $P(\{p_k \in P | k = 1, 2, 3, ..., n\})$. 2)Then we join each f_{p_k} to make all *Friends* $F = f_1 \cup f_2 \cup f_3 \cup ... \cup f_n$ in each urban event. When we carry out this procedure, the number of times to access to Twitter API requires only n times. If we use the existing procedure, *Clique Percolation Method*[82][83], which can detect the cluster and as in theory of graph, we must access to Twitter API more than thousand times to acquire the relationship between one user and another. So if there are n participants in urban event, we must access to Twitter API for $1000n$ times. While CPM is not suitable to analyse in real time considering the server load, our method is suitable. After acquiring the set of friends F in each urban event, we calculate $ratio_{f_i}$ which is the ratio of each *Friends* in F as follows:

$$ratio_{f_i} = \frac{followed}{n} \tag{4.14}$$

We assume the graph is set up with $ratio_f$ as vertical axis and sorted friends descending order as horizontal axis would follows power-low distribution. From this assumption, we use the method used in power-low distribution, regression analysis, and Gini's coefficient calculation, then we attempt to classify urban events.

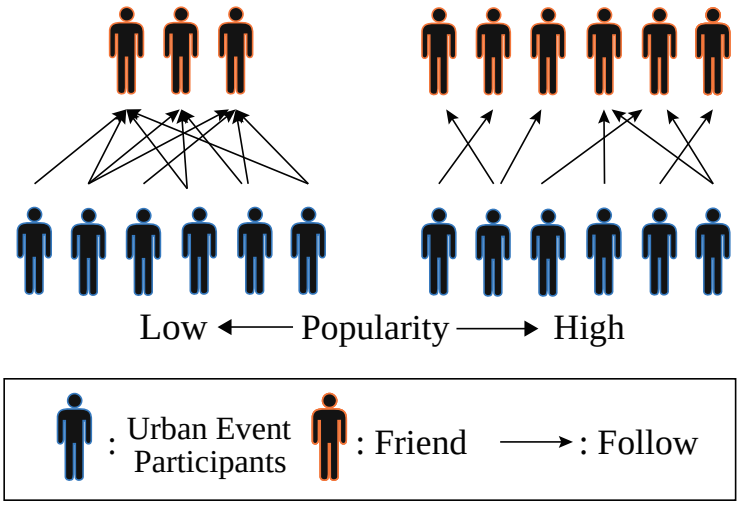


Figure 4.12: The relation between *friends* and *popularity*

Regression Analysis

Using regression analysis, we can obtain the formula of power-low distribution. The procedure of regression analysis is as following; 1)First take the logarithm of the horizontal and vertical axis of the graph in each event and take points in order to make double-logarithmic graph. Since the vertical axis shows friends accounts, we replace the friends as numbers $(1, 2, 3, ..., n)$ to take the logarithm. 2)Then we determine the formula $y = ax + b$ of double-logarithmic graph with a regression analysis. 3)Lastly, we determine the formula $y = e^b x^a$ of power-low distribution. x^a represents the curvature of power-low distribution. The bigger x^a is, the looser the curvature is. When the curvature is tight, the difference between the $ratio_f$ of friends accounts would become large. It indicates that some friends are followed by more participants compared to other friends. To sum up, we can estimate how participants follows friends by calculating of the curvature of power-low distribution.

Gini's Coefficient

Gini's coefficient is a factor used in the power law distribution in economics, and reflects the inequality of income distribution in a population. When we apply Gini's coefficient to our study, it expresses the inequality of following from the participants in urban event. To determine Gini's coefficient *gini*, first we calculate the average *avr* as

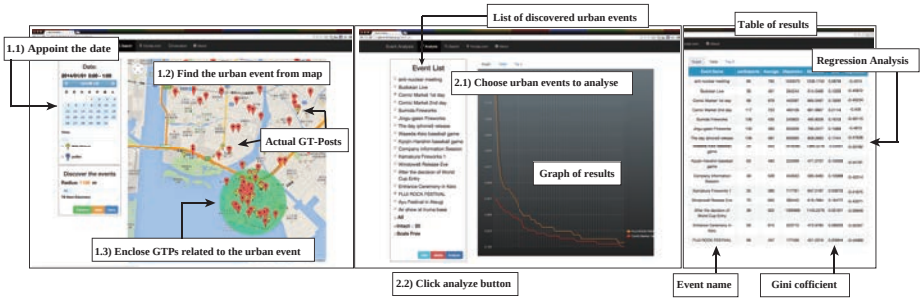


Figure 4.13: Our tool: (1)urban event discovery function (2) urban event Analysis Result in Graph function (3)urban event Analysis Results in Table

Table 4.7: The implement environment

Language	Python・HTML・Javascript・CSS
Framework	Django v1.5.4
OS	Ubuntu 12.04 LTE

follows:

$$avr = \frac{ratio_1 + ratio_2 + ... + ratio_n}{n} \tag{4.15}$$

$$ad = \frac{\Sigma|ratio_i - ratio_j|}{|F|(|F| - 1)} \tag{4.16}$$

$$gini = \frac{ad}{2avr} \tag{4.17}$$

Since *gini* has a range of value $[0, 1]$, the value close to 1 shows the difference between friends is large, and closer to 0 is small. Gini coefficient of each events reflects the following, which friends get from participants of each urban event.

4.5.3 Design and Implementation

We designed and implemented an interactive web-based tool which analyses urban event's *Popularity* based on our proposed method. Figure 4.14 shows the system architecture of the implemented tool and Table 4.7 shows the implementation environment. In order to discover various urban event, *GT-Posts Collection Module* collects *GT-Posts* from Twitter and saves them into SQL databases in every minutes. In this study, we focused on urban events in Japan, so that we collected *GT-Post*s with the parameters of longitude and latitude including whole of Japan through Streaming API[6] (about

[6]https://dev.twitter.com/docs/api/streaming

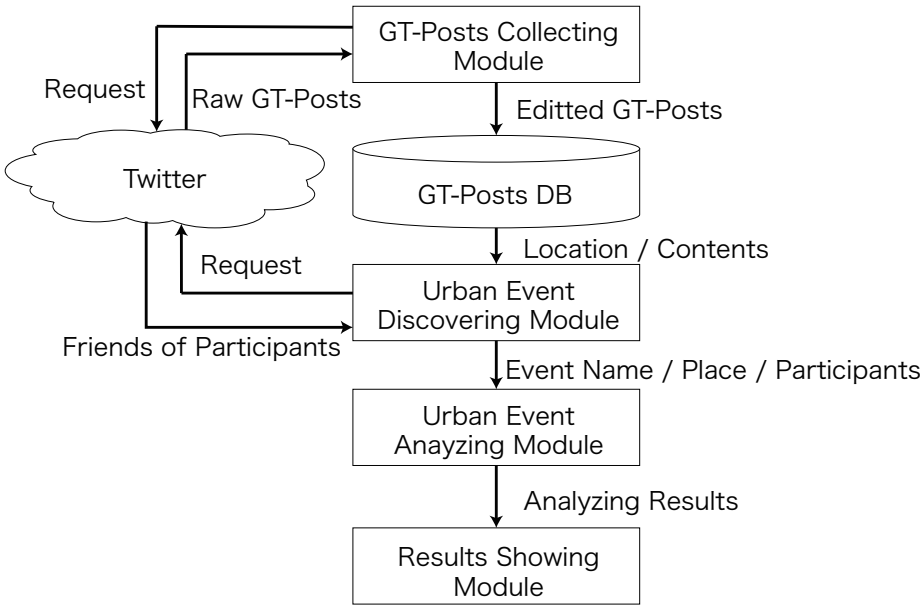


Figure 4.14: System architecture

250,000 posts/day).

Though this thesis focuses on urban event classification, we have to discover urban events as the target of classification analysis. The tool enables users to discover urban events interactively by exploring *GT-Post*s mapped to *Google maps*. When users discover an urban events from *GT-Post*s on *Google maps*, they can enclose the posts by optimal circle where urban event may happened (see Figure 4.13(1)). Then they name and register the urban events for further analysis of classification. After registering the urban events, *Urban Event Analysing Module* analyses the events' *Popularity* automatically. The analysis result of participants's friend information is shown in power-low distribution graph (see Figure 4.13(2)). In addition, the result of *Popularity* classification by regression analysis and Gini's coefficient analysis is shown in the table (see Figure 4.13(3)). Though only the regression analysis and Gini's coefficient are used in our study, we designed the module to be able to extend for other analysis method in the future.

Table 4.8: Target urban event list

Date	urban event name	Participants
2012/02/24	Japan Media Arts Festival	68
2012/02/26	Tokyo Marathon	120
2012/04/04	Keio Entrance Celemony	20
2012/07/25	Kamakura Fireworks Festival	44
2012/07/27	Fuji Rock Festival	68
2012/07/28	Sumida River Fireworks	136
2012/08/04	Atsugi *Ayu* Festival	37
2012/09/15	Tigers-Giants baseball	63
2012/11/01	Perfume *Budo-kan* Live	36
2012/11/03	Iruma Airbase Festival	45
2012/12/09	Job hunting meeting	48
2012/12/29	Comic Market	99
2013/06/01	Waseda-Keio baseball	20
2013/11/23	Tokyo Motor Show	108

Table 4.9: The answerer of *Yahoo! Crowdsourcing*

Total	946
Gender	male: 541 female: 405
Age	10's: 5 20's: 157 30's: 388 40's: 278 50's and over 50's:

4.5.4 Experiment

In this section, we evaluate the validity of our classification method by comparing our analysis results with the ground truth. First, we explain an experiment for obtaining the ground truth. Then, we present the result of comparison between the ground truth and our analysis result. For the evaluation, we used 14 different kinds of urban event. We obtained and regarded the ranking of the popularity of urban event based on general users as ground truth.

4.5.5 Datasets for the Ground Truth

We used 16 types of events which are defined by Japan Event Industry Development Association[7]. Of those 16 types, we made urban event list containing 14 types of urban events which were discovered in our tool (see Table 4.8). *GT-Post*s of Twitter used for making the list were collected from 1st Nov. 2011 to 14th Jun. 2013. If an urban event were held on several days, we applied the day when the largest amount of *GT-Post*s exist. The number of participants differs among different urban events. In order to standardise the number of participants for analysis, we chose 20 participants randomly to fit the smallest number of participants' event, *Waseda-Keio* baseball game (intercollegiate baseball game).

To create the ground truth of *Popularity* for 14 types of urban events, we used *Yahoo! Crowdsourcing* by applying a *Paired Comparison Method* (PCM). PCM is one of the methods that distance scaling in one-dimensional to be compared one to one inspection target. Since our research goal is to apply *Popularity* to the recommendation system, we prepared the question as "Which one is more recommendable to everyone ?". As the method of PCM, we used *Thuston's method of paired comparisons* to lighten the burden of the answerers. We recruited the answerers by using *Yahoo! CrowdSourcing*. In our experiment, we request the users in the crowdsourcing service to answer the $91(= {}_{14}C_2)$ questions. Figure 4.15 shows screenshot of actual question to the users. We collected answers from 946 users shown in Table 4.9. We treat the results of the investigation as the ground truth (see Table 4.10).

The result of popularity analysis based on our proposed methods are shown in Table 4.11 (regression analysis) and Table 4.12 (Gini's coefficient). Table 4.11 is constructed from urban event by name and the value α used in $y = Cx^{\alpha}$ which expresses the curvature of power-low distribution. To determine which result is more similar to the ground truth, we use *Spearman's rank correlation coefficient* shown in Table 4.13. Consequently, the result of regression analysis is similar to the ground truth and it seems to be effective to estimate *Popularity*. High popularity events in the ground truth (table 4.10), such as *Sumida* river festival or Tokyo Marathon, are similarly located as higher position in the result of regression analysis (Table 4.11). In addition, low popularity events in the ground truth are also located in lower position in Table 4.11. However, there are some difference between the ground truth and the result of regression analysis. For example, *Kamakura* fireworks festival is placed in a medium position in regression

[7] http://www.jace.or.jp/

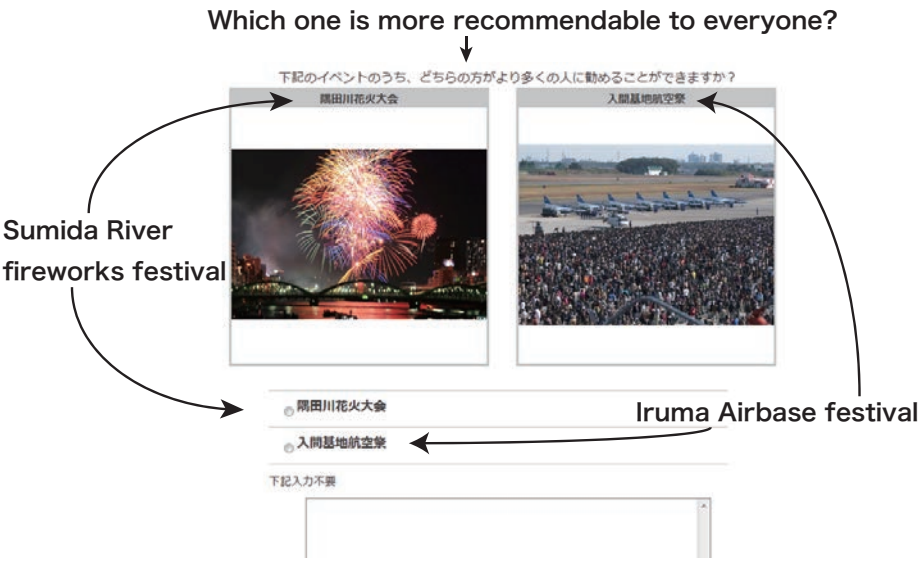


Figure 4.15: The screenshot of actual question in *Yahoo! CrowdSourcing*

analysis. However, it is located in the second place in the ground truth. The reason why it caused is that the word " fireworks" is a popular word and makes the answerers, who do not know the locality of *Kamakura* fireworks festival, consider it to be very popular. In terms of the result of Gini's coefficient, though lower *Popularity* events are similar to the ground truth, higher *Popularity* events are different to the ground truth. In higher *Popularity* events, there are the accounts which are famous (e.g. Son Masayoshi[8], Utada Hikaru[9]) and have more followers than any other accounts. Thus, these accounts are also followed by event's participants and let Gini's coefficient high.

4.5.6 Discussion

Various challenges are addressed for providing further classification of urban events and actual applications. The first challenge is to combine our tool with various techniques for discovering urban event. Current tool needs users to discover urban events manually by exploring *GT-Post*s because this study mainly focused on classification of the discovered events. Combining urban event discovering technique such as [84][85] must support to create full-automatic event recommendation application. The second challenge is improving to detect actual participants to the urban events. Current tool

[8]http://twitter.com/masason
[9]http://twitter.com/utadahikaru

Table 4.10: Ground truth

Rank	urban event name	Value
1	Sumida River Fireworks	1.428
2	Kamakura Fireworks	0.935
3	Fuji Rock Festival	0.376
4	Tokyo Marathon	0.362
5	Tokyo Motor Show	0.306
6	Iruma Airbase Festival	0.134
7	Atsugi *Ayu* Festival	0.120
8	Tigers-Giants baseball	0.011
9	Comic Market	-0.047
10	Perfume *Budo-kan* Live	-0.134
11	Japan Media Arts Festival	-0.160
12	Waseda-Keio baseball	-0.611
13	Job hunting meeting	-1.171
14	Keio Entrance Celemony	-1.55

regarded all users who have *GT-Post*s around the event's place as participants. However, some of users may not participate in the events. Since they become a noise for the classification analysis, we should confirm whether the user really participated to the event or not. To solve this issue, one approach is the content analysis such as the *GT-Post*s' text or pictures. Hiruta et al. [86] applied natural language processing to check the text of *GT-Post*s is related to location and Wang et al. [87] used machine learning to increase the credibility of information. Combining these techniques should provide validate set of participants to the urban events. The third challenge is also combining content analysis to the event classification. Though we focused to analyse friend information of the participants, it is possible to use content information to understand the participants' interest. Nevertheless, since there is a API access limitation to Twitter, we have to consider appropriate way to analyse the contents. The last challenge is to prepare more ground truth for evaluating our method more accurately. Though we used the result of PCM through crowdsourcing as a ground truth, we may also obtain actual participants' information such as statistics of their age, sex or interest from urban events' organizer. This information should be very useful for evaluating the diversity of participants in urban events.

Table 4.11: Regression Analysis

Rank	urban event name	α
1	Sumida River Fireworks	-0.491
2	Tokyo Marathon	-0.471
3	Comic Market	-0.452
4	Fuji Rock Festival	-0.450
5	Kamakura Fireworks	-0.444
6	Tigers-Giants baseball	-0.443
7	Japan Media Arts Festival	-0.431
8	Tokyo Motor Show	-0.4302
9	Atsugi *Ayu* Festival	-0.4295
10	Perfume *Budo-kan* Live	-0.4286
11	Iruma Airbase Festival	-0.427
12	Job hunting meeting	-0.420
13	Keio Entrance Celemony	-0.357
14	Waseda-Keio baseball	-0.338

4.6 Application 2: Transportation Modes Estimation

The second application that using the geolocation data is estimating transportation modes by analyzing moving trajectories that are measured by GPS or Wi-Fi. The trasportation modes are the way that people used to move around the city: buses, taxies, cars or walk. Transportation trajectories are composed with multiple reference points and each of points contains latitude, longitude, and timestamp. When we able to estimate the transportation modes, we can use this method for various mobile applications. For example, we can develop life–logging services that suggest some alternative modes to users. Otherwise, we can use for personal assistant services, such as Siri.

Zheng et.al [6] have calculated velocity and acceleration from the continuous reference points of a moving trajectory and assumed that the transportation modes are switched at the reference point that those values have a drastic change. The moving trajectory between those points that the transportation modes have changed is called as a *segment*. With this assumption, Zheng et.al have reported that the segment can be detected with 90% accuracy (*segment detection tasks*). In this thesis, we tackle the task, *transporation modes estimation tasks*, which is estimating which transportation mode is used in the segment. The overall of transportation modes estimation is depicted in Figure 4.16.

We consider that one transportation mode is assigned to each segment that detected

Table 4.12: Gini's Coefficient

Rank	urban event name	Value
1	Tokyo Motor Show	0.414
2	Perfume *Budo-kan* Live	0.340
3	Tokyo Marathon	0.298
4	Japan Media Arts Festival	0.247
5	Fuji Rock Festival	0.210
6	Comic Market	0.191
7	Kamakura Fireworks	0.183
8	Sumida River Fireworks	0.168
9	Iruma Airbase Festival	0.117
10	Tigers-Giants baseball	0.109
11	Job hunting meetings	0.106
12	Keio Entrance Celemony	0.088
13	Atsugi *Ayu* Festival	0.0390
14	Waseda-Keio baseball	0.037

Table 4.13: The results of Spearman's rank correlation coefficient

Method	Spearman's coefficient
Regression analysis	0.749
Gini's coefficient	0.389

by the segment detection task. Given this assumption, in the transportation modes estimation task, we estimate the modes by using machine learning methods with features extracted from the moving trajectory of each segment. There is a lot of machine learning methods (e.g SVM or decision tree (DT), whose inputs are features that are designed by researchers manually. As well, Zheng et.al have also designed the 'basic features' [6], such as velocity and acceleration, in order to use SVM and DT. Moreover, they have designed the 'advanced features' [7] as shown in Table 4.14. As combining basic features and advanced features, Zheng et.al has reported that the accuracy of the transportation modes estimation has increased. In fact, the features extraction is important to estimate the modes by the machine learning approaches. However, it is difficult to extract the effective features from the moving trajectories due to their less information. Therefore, we assume that there is another features that can be useful to estimate the transportation modes with the higher accuracy.

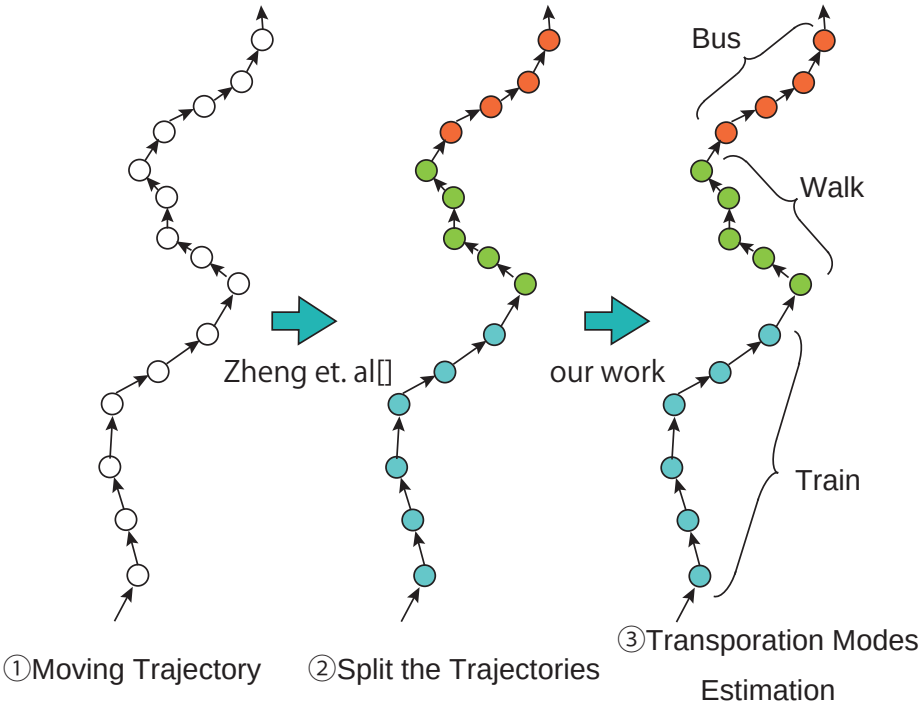


Figure 4.16: The overall of transportation modes estimation. The circle point represents reference point measured by GPS or Wi-Fi. First, segments are detected by analyzing velocity and acceleration of each reference point. Then the segments are assigned to the transporatation modes.

4.6.1 Approach

In order to explore the features that have not used in the Zheng et.al works, we adopt deep neural networks. One of the way to use deep neural networks is features extractor. First, we define the input vector for feature extraction. Thanks to the previous related work, we can assume that not only the velocity, duration, and distance are the significant fundamental factor to discriminate the transportation modes, but also the combination of them. Given the N reference points $\{p_1, p_2, \ldots, p_n\}$, which are contained in same segment and represented as $p_i = \{latitude, longitude, timestamp\}$, we then calculate a three dimensional vector, $gps_i = (d_i, t_i, v_i)$; the distance d_i, duration t_i, velocity v_i between the points p_i and p_{i+1}. Hence, one segment that contains the $m + 1$ reference points is represented as the sequence $\{gps_1, gps_2, \ldots, gps_m$ and used for the input data for the neural networks.

Table 4.14: Basic and Advanced Features [6][7]

Category	Feature
basic feature [6] 10 dimension	The length of a segment Average velocity Velocity Expectation Velocity Variance Top 3 of velocity Top 3 of acceleration
advanced feature [7] 3 dimension	The change rate of velocity Stopping rate The change rate of direction

However, the input data is sequential, we cannot input it to traditional neural networks. One of the traditional neural networks, for example, is autoencoder. Autoencoder is an unsupervised model that compresses the dimension of input data to the less dimension, which contains the information of input data. Autoencoders and other variants, such as models consisted of multiple autoencoders stacked, called stacked autoencoder [88], require the fixed dimensional input data during the mini-batch training. The data to estimate the transportation modes is sequential data as mentioned above, we cannot adopt those models.

To remedy this problem, we adopt recursive autoencoders (RAEs) [89][90]. RAE is proposed by Socher et.al [89] to use in the natural language processing research domain. In the NLP research, the number of tokens that is contained in the textual data (e.g. a word or a sentence) differs in various tasks. Socher et.al utilize RAEs to NLP tasks to treat the sequential textual data. In short, RAEs are suitable for the transportation modes estimation, for the segment data is represented as the sequential data.

We show the overall of applying the RAE to the transportation modes estimation task in Figure 4.17. RAEs are based on binary trees architecture and we represent it as $(p \rightarrow c_1, c2)$. With reference to figref, we can represent the trees:

$$\{(GPS_1 \rightarrow gps_1, gps_2), (GPS_2 \rightarrow gps_3, gps_4), (GPS_3 \rightarrow gps_1, gps_2)\}, \tag{4.18}$$

where GPS_i represent the hidden layers and they have same dimention as well as the input. Simultaneously, GPS_i is calucalated as follows:

$$p = f(W^{(1)}[c_1; c_2] + b^{(1)}), \tag{4.19}$$

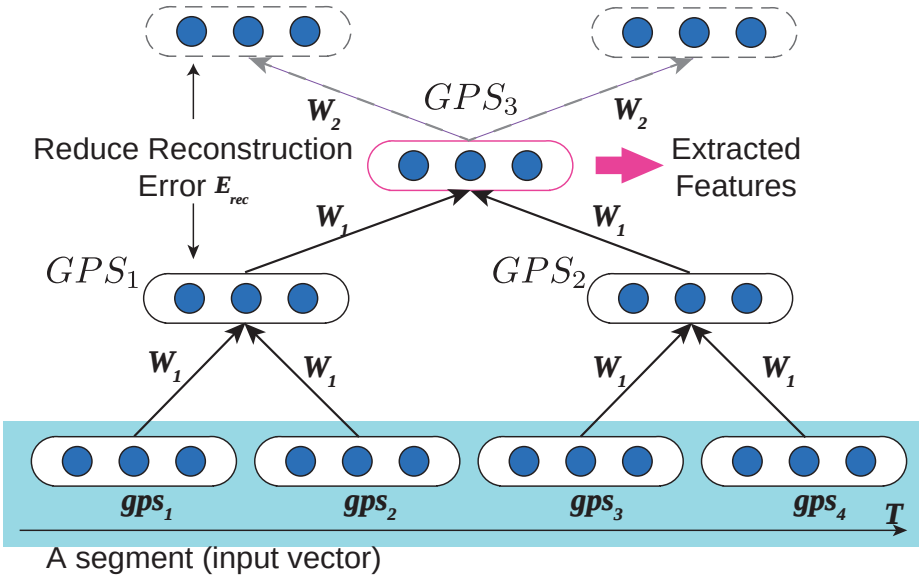


Figure 4.17: Recursive autoencoder.

where $W^{(1)} \in \mathcal{R}^{d\times 2d}$ is the weight parameters and d is the dimension of the input vector; $d = 3$ in our approach. Moreover, $b^{(1)} \in \mathcal{R}^d$ represent the bias parameters. We use *tanh* or *sigmoid* for activation function f.

In order to evaluate the dimension is compressed whether effectively or not, then, we calculate the reconstraction error between the reconstracted vector $[c_1'; c_2']$ and the input vector. The reconstracted vector is calculated as:

$$[c_1'; c_2'] = f(W^{(2)}[c_1; c_2] + b^{(2)}), \tag{4.20}$$

where $W^{(2)} \in \mathcal{R}^{2d\times d}$ is the reconstraction weight parameters. The reconstraction error is:

$$E_{recon} = \frac{1}{2}\|[c_1'; c_2'] - [c_1; c_2]\|_2^2. \tag{4.21}$$

RAE is trained to minizie E_{recon} by optimizing W and b. To determine the tree architecuture, we define a function $T(y)$ that returns the the binary tree which minimize the reconstruction error. Simultaneously, we decide the architecture of $RAE(gps)$:

$$RAE(gps) = \arg\min_{y\in A(gps)} \sum_{s\in T(y)} E_{rec}([c1; c2]_s). \tag{4.22}$$

Applying the RAE at each segment, the $3m$ dimensional input vecotor of segment is compressed to 3 dimension. This is the problem that the information is lost when

the number of reference points in the segment is large. Moreover, since RAE is an unsupervised learning model, the features is not represent the transporatation mode sufficiently.

To remedy these problems, we propose n-transformed supervised recursive autoencoder (NTS-RAE), which is extended RAE to supervised learning model and can adjust the compressed dimension arbitrary. The two extention is as follows:

Supervised Learning.

To extract the features that represent the characteristic features for transportation modes, we train the RAE with the ground truth labels. Related to our extention, while Socher et.al proposed the supervised learning with all vectors p, we train the NTS-RAE with only the last layer. The supervised training is conducted as follows:

$$E_{CE}(p,t) = -\sum_{k=1} t_k \log d_k(p), \tag{4.23}$$

$$d(p) = softmax(W^{label}p), \tag{4.24}$$

where E_{CE} is the cross entropy loss function and t_k is the k-th ground truth labels which is represented as one-hot vector. Given the features p, d is the categorical probability distribution of the transportation modes. d is K dimensional vector, K is the number of kind of transportation modes and d_k is the probability of transportation mode k. Note that $\sum_k^K d_k = 1$ because of probability distribution.

Compressed Dimension Adjustment.

Given a segment of a GPS trajectory, gps_i is the feature that the information of previous features($\{gps_1, \ldots, gps_{i-1}\}$) is compressed. We fix the subset tree ($GPS_i \rightarrow GPS_{i=1}, gps_i$) as shown in Figure 4.18. Simultaneously, we initialize the first input vector gps_0 by random sampling. The dimension of this gps_0 is n and we can adjust the dimension of features by this gps_0 input vector. Therefore, we can get the good representation features by this adjustment.

To summarize with these extension, NTS-RAE is trained to minize the regulaized objective function:

$$E([c_1; c_2], p, t) = (1-\alpha)E_{rec}([c1; c2]) + \alpha E_{CE}(p,t). \tag{4.25}$$

Note that the α is the hyper-parameter and we determined through the experiment. The $\alpha = 0$ means let NTS-RAE be an unsupervised learning model.

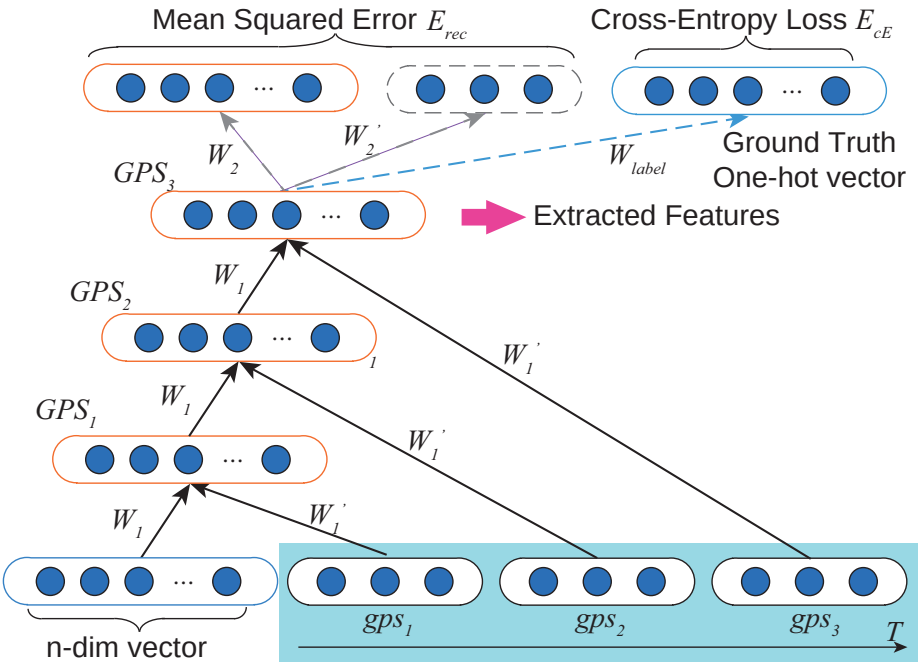


Figure 4.18: N-Transformed supervised recursive autoencoder.

4.6.2 Experiment

To evaluate our proposed model, NTS-RAE, we conduct two experiment: 1) accuracy comparison and 2) accuracy validation with additional extension.

Dataset.

In our experiment, we use Microsoft GeoLife dataset [91]. GeoLife dataset consists of the GPS moving trajectory gathered from 65 users for 10 months. The dataset contains 9111 of segments. Simultaneously, the segments have labels that indicated the 11 kinds of transportation modes as shown in Table 4.16. In our experiment, we divided the GeoLife dataset into 3 subsets: training, validation and test datasets. Note that these subsets were divided as training : validation : test = 7 : 1 : 2. We trained the NTS-RAE with training dataset, and then validated and determined the hyper-parameters by comparing the accuracy of NTS-RAE with validation dataset. Finally, we evaluated the model with those hyper-parameters.

The accuracy of each features.

In order to evaluate our NTS-RAE whether it can extract the effective features, we compared with an accuracy with the previous work:

- basic feature [6]
- feature extracted by our NTS-RAE
- combination of basic feature and NTS-RAE feature.

The accuracy is calculated as follow:

$$\text{acc.} = \frac{TP}{TP + TN}, \tag{4.26}$$

where each notion denotes:

- TP: the number of output that estimate the transportation modes correctly
- TN: the number of all data.

We used three traditional supervised machine learning methods to classify the segments into the 11 transportation modes: SVM, decision tree and logistic regression. Note that we determined the kernel of SVM as RBF kernel (gaussian kernel) and other hyperparamters are determined by grid search.

The results are shown in Table 4.15 and Table 4.16. Table 4.15 shows the accuracy of all transportation modes. As a result, while the accuracy of the decision tree with only the basic features [6] reached 75%, combining the features extracted by our NTS-RAE achieved 76.6% that is 1.6% higher. Simultaneously, the results of other machine learning methods shows as well. In summary, our NTS-RAE can extract the effective features for transportation modes estimation which is not contained in basic features.

Table 4.16 shows the accuracy of each transportation mode respectively. In this experiment, we adopted decision tree because it has a good performance at the Table 4.15. With our NTS-RAE features, the accuracy of *taxi/car* became 3% higher than with only the basic feature. This means that NTS-RAE extracted the effective features of taxi/car from the data. Moreover, the accuracy of the other transportation modes also became higher as well.

At the bottom of Table 4.16, we calculated the micro average that indicates the average of the accuracy of each transportation mode and the macro average that indicates the accuracy of all segments. As a result, both average score shows that combining

Table 4.15: Accuracy of Microsoft GeoLife Datasets

Data	Classifier		
	SVM	Dicision Tree	Logistic Regression
Manual Feature[6]	45.3%	75.0%	65.0%
NTS-RAE Feature	48.9%	44.8%	50.1%
Manual + NTS-RAE Feature	**55.6%**	**76.6%**	**69.9%**

Table 4.16: Accuracy of Each Transportation Modes

Transportation mode		Feature	
Name	Data	Manual Feature	Manual + NTS-RAE
Taxi	497	32%	**35%**
Walk	3749	88%	**89%**
Bus	1800	66%	**68%**
Car	774	64%	**67%**
Bike	1540	78%	**79%**
Subway	571	62%	**65%**
Train	154	67%	**80%**
Airplane	14	50%	**100%**
Run	3	0%	0%
Boat	7	0%	0%
Motorcycle	2	0%	0%
Micro Average	Total 9111	75%	**77%**
Macro Average		46%	**53%**

the basic features and NTS-RAE features is better to estimate the transportation modes than using only the basic features.

Extension evaluation.

We also evaluate whether our two proposed extensions are effective or not for the transportation modes estimation[10].

- **Supervised Learning**. We validated how the accuracy changes by the supervised learning. To validate the effect of the supervised learning, we change the value of hyper-parameter $\alpha = \{0.001, 0.5, 1.0\}$. The result is depicted in Figure 4.19. As a result, the accuracy became high as α increased. This result means

[10]In this experiment, we did not conduct the grid search and fixed the value of the parameters as same as previous experiment.

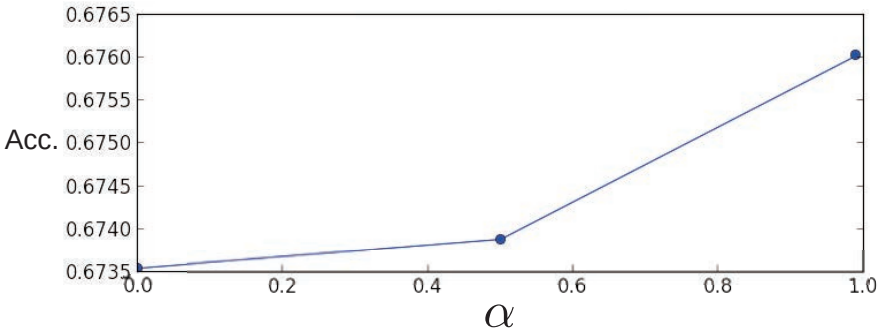


Figure 4.19: Supervised Learning Effect.

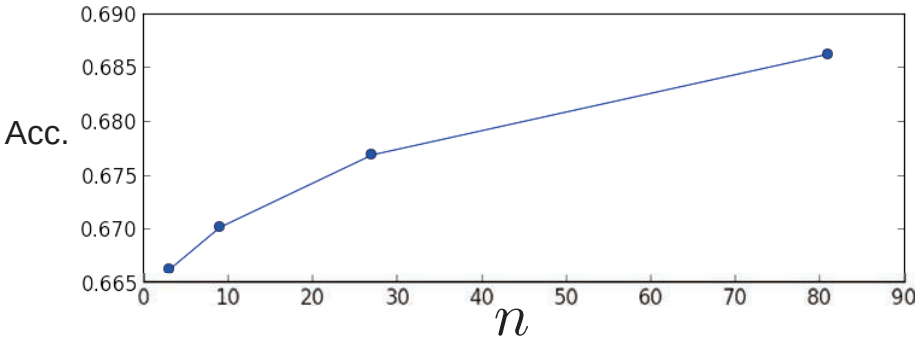


Figure 4.20: Compressed Dimension Adjustment.

that the supervised learning is effective to extract the features for the transportation modes estimation.

- **Compressed Dimension Adjustment**. In order to validate the effect of the compressed dimension adjustment, we changed the dimension $n = \{3, 9, 27, 81\}$ and calculated the accuracy. The result is shown in Table 4.20. The accuracy became higher as the dimension became high. This result shows that high dimensional features is better to estimate the transportation modes as higher.

4.6.3 Discussion

As results of the experiment, we confirmed that NTS-RAE is able to extract the features that is effective for the transportation modes estimation. Simultaneously, we demon-

strated the two proposed extensions are also effective. In this work, while we only validate those extensions with the limited value range of hyperparamters, we have to explore the other values which can estimate the transportation modes with much higher accuracy. Moreover, Socher et.al [89] have reported that the semi-supervised learning can increase the performance of models because the semi-supervised learning enables us to use unlabeled data. Hence, extending NTS-RAE to semi-supervised learning is one of our future work.

4.7 Discussion

In conventional urban computing, the data utilized in their studies is only gathered by conducting the experimental participatory sensing, or collected from the location-based social networks, that is, the geospatial information is attached to the data explicitly. Hence, the data which does not contain the geospatial information is ignored. With our GeospaceMapping, we can reuse the non-geospatial data even though it can estimate the location only in the rough granularity; only the prefecture in Japan. We are not able to use those reusable urban non-geospatial data for urban computing actively, yet the city are able to filter the data which is not related to their prefecture passively. This is a significant step forward in increasing the amount of urban data. Simultaneously, we can regard GeospaceMapping as a pre-processing that is geolocation attachment and is similar to that ZCA whitening in a computer vision domain or stop words removal in a natural language processing domain. Our pre-processing enables urban computing to use much more useful data that indicates the status of the city.

In terms of requirements of expert knowledge, using deep learning release us from describing the expert knowledge. In our GeospaceMapping and the last application as examples, thanks to deep learning approaches, we do not need to consider which terms are efficient to estimate locations or design the features to estimate the transportation modes. We can say that with deep learning, sensing the status of city becomes more affordable than conventioanal approaches (e.g. feature engineering).

Regarding the accuracy of GeospaceMapping, acc@1= 37 % is still inefficient to regard the text data such as non-geotagged tweets as location-based text data. However, if expand the candidate of the result into three candidates, acc@3 becomes 50 %, that is, it would become useful for eliminating the noise (non-target prefectures). After all, we need to explore the proper architecture of models and increase the amount of dastaset to achieve higher accuracy.

4.8 Summary

This chapter has focused on how can we facilitatively understand the city context with the geolocation. Gathering information from users, especially citizens, we can know what happens in a city. However, due to privacy concerns, there is scarcely the information that indicates the location. To tackle this issue, we proposed the method to estimate the location that the text and/or images indicate by using deep neural networks. In addition, in order to present the importance of geolocation, we demonstrated the two kinds of application that can be deployed when we get a lot of geolocation data. The first application is that estimating the popularity of city events by analyzing the geo-tagged tweets and followees of users who participated the events and posted about it. The second application is that estimating the transportation modes which citizens used to move around the city by using the features extracted from the sequence of geolocation information by using the variants of recursive autoencoder which we proposed. With our GeospaceMapping, we can obtain the geolocation information that facilitates us to understand what happens in the city. Simultaneously, GeospaceMapping does not require us domain knowledge in terms of the words indicating the place or the way to mix the multimodal data: texts and images thanks to deep learning. In addition, our GeospaceMapping help us to gather the geospatial-data from users without providing the desighnated participatory sensing and also compelling users (citizens) to offer the data with their location explicitly. GeospaceMapping can be regarded as the preprocessing that tranfer the various existing data such as tweets, searching history, or the log of call center to geo-spatial urban data. After this transfering, we can use the urban data for understanding the city with reference to long-term periods.

Chapter 5

CityInspector: Urban Recycled Sensing

5.1 Introduction

This chapter describes urban recycled sensing that recycles un-used (or already used) data for urban computing. In order to realize this sensing, we propose a package of edge devices combining existing cameras with edge computers. In this thesis, we focus on road inspection as an example subject.

The road is one of the most important infrastructures of a city in planning and development. For instance, people usually use them for going somewhere or for planning land utilization to enrich their livelihoods. However, many roads need repairing since most of them are built in periods of rapid economic growth and have been deteriorating since. Thus, to inspect their condition for road repair, the city administration needs to employ people for constant inspection. Yet manual road inspection is expensive and takes a lot of time; for instance, in order to detect the damage or blur of road markings, people have to check by eye, whose ability has certain limits. In addition, in certain regions such as Japan, public funds for road inspection have been reduced due to current societal conditions. In short, manual road inspection and repair is not enough for sufficient maintenance. Most previous work has therefore focused on making the cost of road inspection cheaper to increase sustainability. Some works have focused on road flatteness[92, 93, 94], potholes[95][96], and cracks[97]. In contrast, we aim to detect the damage or blur of white lines. To our knowledge, only our previous work addresses this problem[98]. Detecting the damage of white lines is difficult to do using smartphone accelerometers such as [92, 93, 94]. Thus, we use a camera to take pictures/videos [97][98]. If we use participatory sensing[99] as well as [97] and collect the images, however, the cost issue still remains due to the cost of platform introduction and labor.

To tackle this issue, we focus on city vehicles, especially garbage trucks. Garbage trucks run their services every day and cover a whole area of the city. For example, at Fujisawa city in Japan, the garbage trucks cover about 65% of 100m mesh grids if they runs in the city about three days [100]. Therefore, if the garbage truck equips a camera and takes pictures of roads, we can obtain road images from the whole area. Furthermore, we do not have to pay additional costs for labor or facilities. However, the number of running garbage trucks is so large (e.g., hundreds of trucks) that it is troublesome to storage and manage image data in a centralized way. Simultaneously, if we upload an image every time a camera takes pictures, it would take great communication costs and bandwidth. In summary, our goal is proposing a system that can be

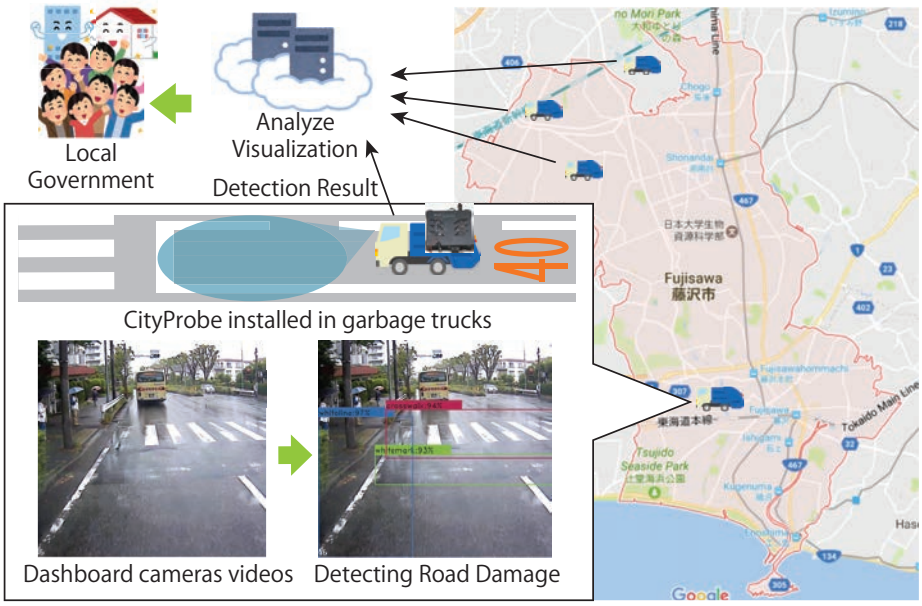


Figure 5.1: System overview. Each garbage truck running in the city detects road markings blur. The cloud computer aggregates and analyzes the detect results. Finally, the system feedbacks them to citizens and administrators.

attached to garbage trucks and detect white line damage on the spot.

In order to validate whether neural networks can *recognize* the status of object, we proposed the road damage decision system, *deep on edge* (DoE) that execute on the edge computer, such as Raspberry Pi 3. The model working on DoE is a very small convolutional neural network and classify the given image of lane marking into two classes; damaged or non-damaged.

While DoE succeeded classification with very high accuracy, DoE is not useful enough to be deployed to garbage trucks because of their requirement that the camera has to be set so as to take pictures towards grounds. Then we use drive recorder cameras that are already set to almost all of garbage trucks. The example of an image taken from the drive recorder is shown in Figure 5.5. In Figure 5.5, we can see not only the lane markings but also other markings, for instance, crosswalk marking. In this study, therefore, we adopt the real-time object detection approach to treat these marks altogether. While there is enormous approach to detect objects, we use vanilla you only look once (YOLO) [101] model and SSD model [102] for the first trial which are

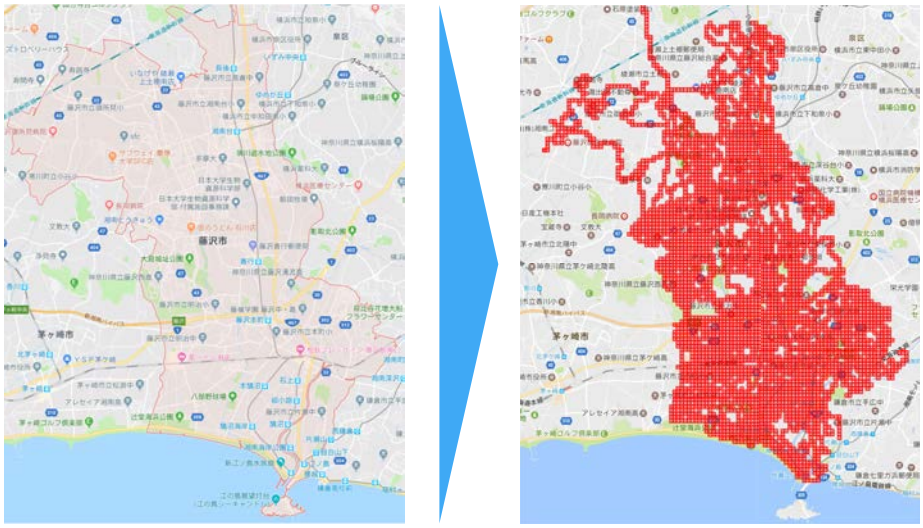


Figure 5.2: The coverage where garbage trucks run. Yin et al. [100] have reported that the garbage trucks cover the 65% area of Fujisawa city in Japan.

based on neural networks. To train the model, we collect the image data from drive recorder of sanitation engineer. Then we annotate the images: surrounding the area of road marking blur with a bounding box. In our experiments with this dataset, we verify whether YOLO models detect the road markings blur. As a result, while YOLO models are not good at mean average precision (mAP) score, the result of blur detection seems to be good with visualization. Therefore, our results may help the administration to know which road to repair in the city and the data of this road damage is able to be used in urban computing.

5.2 Requirements

In this section, we indicate requirements of CityInspector in order to facilitate the road damage inspection by piggybacking the garbage trucks.

5.2.1 Road Damage Detection

CityInspector needs to inspect the road to find the damage from the camera installed in garbage trucks. In order to satisfy this requirement, image processing technology is suitable. One of the image processing technology, a deep neural network is very useful because of its high performance. While deep neural networks can discriminate the

images into 1000 classes [72], which contain various object images, there is no study that discriminates the status of objects (i.e. our target task, road damage) to our knowledge. Therefore, firstly, we needs to investigate whether the deep neural networks can discriminate the status of objects. After the investigation, then, CityInspector adopt the deep neural networks to inspect the road damage.

5.2.2 Detection Environment and Data Management

The second requirement is the edge computer environment to detect road damage. In general, a lot of application that analyzes the images or videos send them to cloud servers because the computational cost of image processing is very high. However, since the dozens of garbage trucks travel through a city, the amount of video data becomes enormous and the communication cost of sending the data to cloud servers is huge. Thus, it is not reasonable to send all video data. Contrary to this, there is a way that preserves the video data in local storage such as SD cards or USB memory. If the video data is preserved in the local storage, it is very troublesome to collect the data. This is because the additional workflow to deliver the storage to somewhere and salvage the data from it, or new network environment, such as Wi-Fi routers, is required.

Additionally, there is another problem related to the data. The information that is very related to the individuals, such as faces and registration numbers, is often in the videos from dashboard cameras. Hence, it is difficult for local government or city administrative to preserve these videos due to the privacy protection ordinance.

To address these issues, CityInspector is required to conduct the road damage inspection on a local environment, a so-called *edge computer*, and not to preserve the video data. Simultaneously, the data to be sent is only the result of the detected damaged road. When the CityInspector conduct the road inspection, it is also required that there is no omission of the road inspection. Therefore, the CityInspector needs to conduct the road inspection within a short time. If we assume that garbage trucks usually runs by $30km/h$ (= $8.3m/s$) and the range of the road being in a video frame is $5m\tilde{1}0m$, the CityInspector needs to process with in 1 second.

5.2.3 Power Management

Given the previous requirement, edge environment, CityInspector needs to be supplied a electric power from the garbage truck. In order not to add a new workflow, the power

Table 5.1: State Transition Table of CityInspector.

State	Engine ON	Engine OFF	After a certain period of time
Inactive	Active	–	–
Active	–	CountDown	–
CountDown	Active	–	Inactive

management should be done by CityInspector self. Therefore, CityInspector needs to start the road inspection with the engine of garbage trucks simultaneously, since the electric power is not supplied before the engine starts. In addition, it is revealed that city officers sometimes take a rest in a daily work and stop the engine for few minutes. It is very inefficient that turning off the CityInspector at the every time of the engine being stopped. To remedy this, CityInspector needs to be turned off in few minutes after the engine stopped. The status transition tables is depicted in Table 5.1 and the status transition diagram is depicted in Figure 5.9.

5.2.4 Result Visualization

Finally, we need to present the results of the road damage inspection where the damaged road exists in the map. Depending on the target of the road damage, the organization to show the result becomes different. For example, while police takes charge of many road markings because road markings hold legal constraints, the local governments take charge road damage such as cracking or pot-hole. Indeed, we need to show the results of the road damage inspection not only all but also filtered with a distinction of organizations.

5.3 First Study: Road Damage Recognition

In order to achieve our goal, we introduce Deep on Edge (DoE), which integrates edge computing and deep neural networks. The overview of our system is depicted in Figure 5.1. DoE consists of an edge computer (e.g., Raspberry Pi 3) with a camera to be attached to garbage trucks. When DoE detects line damage, the results are reported and sent to cloud computing. Then, we use those reports and understand city condition. We treat the task of line damage detection as a classification problem. We train a convolutional neural network (CNN), a type of deep neural network, on labeled images

Table 5.2: Road Damage Benchmark Dataset.

Class	Type	Undamaged	Damaged	No line	Total
Binary	Train	10696	14304	–	25000
	Test	3829	5073	–	8902
Trinary	Train	15445	11568	7987	35000
	Test	3932	2957	2013	8902

on a GPU server. At inference time, DoE is loaded on an edge computer and outputs a discrete probability distribution, assigning each image a likelihood that the white line in the image is damaged. There are some constraints to using DoE on edge computers due to restricted performance, while on the other hand we do not have to consider the number of parameters or inference speed when we use DoE on a high-performance computer. So to use DoE on edge computers, we design the CNN architecture to be as small as possible but to keep the accuracy high. To evaluate DoE, we compared it with baselines on the line damage detection task. As a result, DoE outperforms baselines on this task while reducing the number of parameters. Simultaneously, we visualize DoE activation so that we can understand how it has learned to detect line damage.

5.3.1 Dataset: Road Damage Benchmark

In this study, for detecting road damage, we focus on the damage or blur of white lines, which we assume is the most common type of lane marking. To collect line images, we attached a normal camera, which can film by 60 fps, on a side of a passenger car so that the camera always films the line. Then, we drove the car within 50km/h for four days from March 30th, 2016 to April 2nd, 2016 in daytime. Note that it was sunny days. While we got videos in which each frame is 1024×768 pixels after filming, we randomly cropped frames into 224×224 pixels. This cropping was done for reducing the training time until the model convergence and allowing the model focus on the line damage. One participant annotated those cropped images with three kind of labels; damaged line, undamaged line, and no line. After the preprocessing described above, we obtained 43000 images of lines. At our experiments, we divide the dataset to 35000:8902. The examples of our dataset are shown in Figure 5.3 and described in detail in Table 5.2.

We pose the task of line damage inspection as a classification problem. For this, we use a dataset of images of lines with three kinds of labels described in the previous

Figure 5.3: Dataset samples. Each image is 224×224 pixels by random clipping from video frames, respectively. The images at top row show damaged line, the images at mid row undamaged and at bottom row no line.

section. The input to DoE are image pixels and the target output is a one-hot vector encoding those labels. Given an image, the output of this model is a probability distribution describing the extent of road damage. The advantage of outputting a probability distribution is that this gives the model the ability to give specific scores to a line image, taking out the necessity of a human expert to give specific ratings.

5.3.2 Deep on Edge: Small Convolutional Neural Network

In order to recognize line damage from images, we adopt a convolutional neural network, which is a special type of feedforward neural network or multi-layer perceptron and works well with two-dimensional images. We design our CNN by referring to the VGG16 architecture[72]. VGG16 is one of the major CNN architectures which was used to win the ILSVR competition in 2014, although it has been outperformed by

great advances such as Inception[73] and ResNet[103][104]. VGG16 only uses convolutional layers with 3×3 kernels and pooling layers with 2×2 kernels. This feature is very significant for DoE since the size of the model is required to become as small as possible to work on edge computers. Given an input image $\mathbf{X}$ of width w, height h and c color channels (usually RGB channels) represented as $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$ at each convolutional layer, it is convolved with d sets of local kernels $\mathbf{W} \in \mathbb{R}^{w \times h \times c \times d}$ and bias $\mathbf{b} \in \mathbb{R}^{d}$ is added:

$$h = \phi(\mathbf{W} * \mathbf{X} + \mathbf{b}), \tag{5.1}$$

where $*$ denotes a convolution operation and ϕ is a non-linear function that we use the rectified linear unit (ReLU, $\max\{0, x\}$). Max-pooling, a form of non-linear downsampling, is applied to the output of the convolution. Max-pooling partitions the input into a set of non-overlapping rectangles by the kernels and outputs the maximum value in each sub-region respectively. This operation is very useful because it reduces the dimensionality of a high-dimensional (high-resolution) output of the convolutional layer and summarizes the activations of neighborhood features so that model becomes robust to local perturbations. Since our input images are filmed from a driven car, the location of lines in the image are not fixed. DoE is built by several alternating convolution layers and max-pooling layers.

In VGG16, the output after some convolution layers and max-pooling layers is flattened for the input of to the following layers, which are fully-connected. If the shape of the output of convolution is $\mathbb{R}^{w \times h \times c}$ and the output dimension of the next fully-connected layer is d, the number of parameters in that FC layer becomes $w*h*c*d$. This is a problem when the size of the input image is large, since the larger the image size is, the larger the number of parameters becomes. To avoid the increase in number of parameters, we use global average pooling[105] instead of flattening. Applying global average pooling allows the number of parameters in the FC layer $c*d$ to be independent of the input image size. At the last layer of DoE, the output is a probability distribution over the possible conditions of the road.

The model of the DoE architecture which we used in our experiment is depicted in Figure 5.4.

5.3.3 Image devision for Practical Use

While DoE is trained with the road images of size 224×224, the size of images from a video camera is much bigger than that. Although our DoE model can take any image

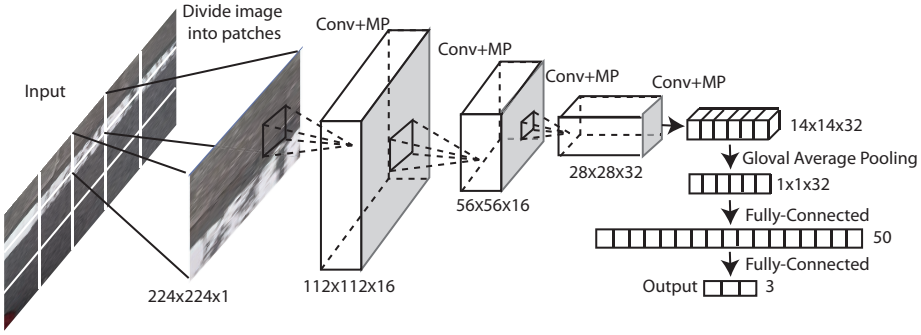


Figure 5.4: Our model on DoE architecture. "Conv + MP" denotes that convolutional operation with 3×3 kernels with strides 1 and max pooling operation with 2×2 kernels with strides 2. After four convolution and pooling, applying global average pooling[105].

resolution, our preliminary experiment showed that DoE cannot detect the line damage accurately at any resolution. In order to tackle this issue and use DoE for practical use, we implement a module that divides the input image into 224×224 sub-regions and reshapes these sub-regions to $X \in \mathbb{R}^{n \times 224 \times 224}$ where n is the number of sub-regions. Even if n is very large, DoE is able to process it all at once. For instance, if the size of an input image is 1280×720, the number of sub-regions becomes $(1280/224) * (720/224) \approx 15$ and the input to model $X \in \mathbb{R}^{15 \times 224 \times 224}$. Although our module crops out the top, bottom, and right sides of the image, this is not a problem because of two reasons: (a) the top and bottom sides of the image usually does not contain the road (b) the road contained on right side is contained in the next input image. Figure 5.4 also shows this module.

5.4 Second Study: Road Damage Detection

In this section, we introduce the road marking blur detection task that we tackled in the thesis.

In developed countries, such as Japan, the roads have been constructed during a period of economic growth all at once. Therefore, a lot of roads becomes deteriorate in nowadays. Needless to say, road markings have become deteriorate too. Since the road markings are very important from the perspective of traffic safety, government and administration dispatch people for inspection to cope with this road deterioration. However, these labor costs become expensive as the area of inspection become wide

Table 5.3: Road Damage Visual Object Class Dataset.

Type	White line	White mark	Color line	Color mark	Crosswalk	Guardrail	Total
Train	1807	963	726	249	37	209	709
Test	1821	1065	635	137	55	424	588

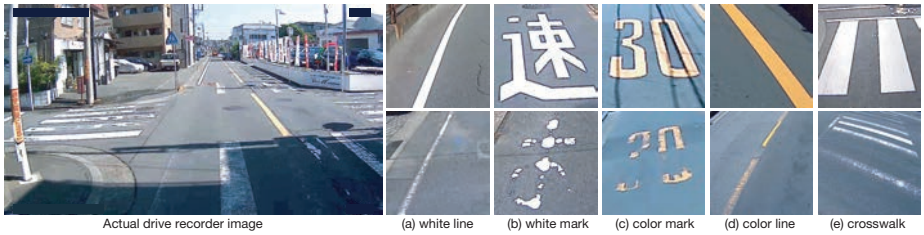


Figure 5.5: Actual drive recorder image and blur situation. (left) The actual drive recorder image. (right) The top row shows NOT blurred road markings. The bottom row shows blurred road markings. (a) The white line as known as lane marking. (b) The white mark showing some message, such as “stop”. (c) The color mark (e.g. a speed limit). (d) The color line, the other type of white line. (e) The crosswalk marking for pedestrians.

and the implementation periods long. In order to solve this problem, we focus on the garbage trucks that they run the whole roads of cities every day. If the garbage trucks are equipped with a camera and film the roads, we can know the condition of roads every day. To examine whether the blur of markings can be detected via a camera on a garbage truck, we restricted the target road marking to only road lane markings (Figure 5.5 (a)) which is the simplest marking, in the previous section. On the other hand, with respect to road marking blurs, there are a lot of kinds and shown in Figure 5.5. In this study, we treat not only road lane markings but also other road markings (e.g. crosswalk) since we use drive recorder images.

5.4.1 Road Damage Visual Object Class Dataset

In this study, for detecting blurred road marking by drive recorder, we created a dataset of road images, as we called *RoadDamageVOC*. To collect road marking images, we obtained videos of drive recorder from three garbage trunks. The videos were shot at 18 fps and filmed for five hours each in daytime on May 10th and 11th, 2017. Note that it was rainy on May 10th and sunny on May 11th. We got videos in which each

frame is 1280×720 pixels and extracted images every 25 frames from the videos. The dataset contains five types of annotation, depending on five kinds of blurred road markings such as white line, color line, and cross walk. Our image dataset has an additional annotation. In terms of shape and color, guardrails are similar to white lines. Thus, we also annotated guardrails in order to distinguish between white lines and guardrails. Before conducting an experiment, we divided the dataset into eight to two in chronological order. We used the former dataset for training our model, and the latter used for a test. The examples of our dataset are shown in Figure 5.3 and in Table 5.3.

5.4.2 You Only Look Once Model

One of the method to detect the damaged road markings that we adopt, is you only look once (YOLO) [101] model, which is the first approach to detect the object using only a single neural network and can be optimized end-to-end directly on detection performance. One of the great features of YOLO is their very fast object detection. Since road markings blur detection requires detecting as fast as possible due to edge processing, YOLO is suited for our work. In the next section, we introduce the overview of YOLO. YOLO is a convolutional neural network for object detection. YOLO begins with a series of convolutional layers and maxpool layers which increasingly downsample the feature maps. The downsampled feature maps are then flattened to a vector, passed through one fully connected layer, and then reshaped once again into a $S \times S \times (C+5*B)$ tensor. The tensor represents the output, which divides up the image representation into a grid of S by S cells. Each cell is responsible for B bounding boxes; each bounding box is represented by 5 scalars: (x, y, w, h) and a confidence score which represents how certain it is that the box contains an object. Each cell also predicts a class, which is a categorical distribution of C classes. In our work, $S = 7$, $B = 2$, $C = 6$. Input images are always resized to 448×448. During training, each ground truth is matched with a predictor box with the highest intersection over union (IOU), so that the predictor box is "responsible" for predicting the object accurately. This allows each box (even in the same cell) to be specialized to certain sizes, aspect ratios, or classes; this improves overall recall. The loss is composed of multiple terms:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{blur} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{blur} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

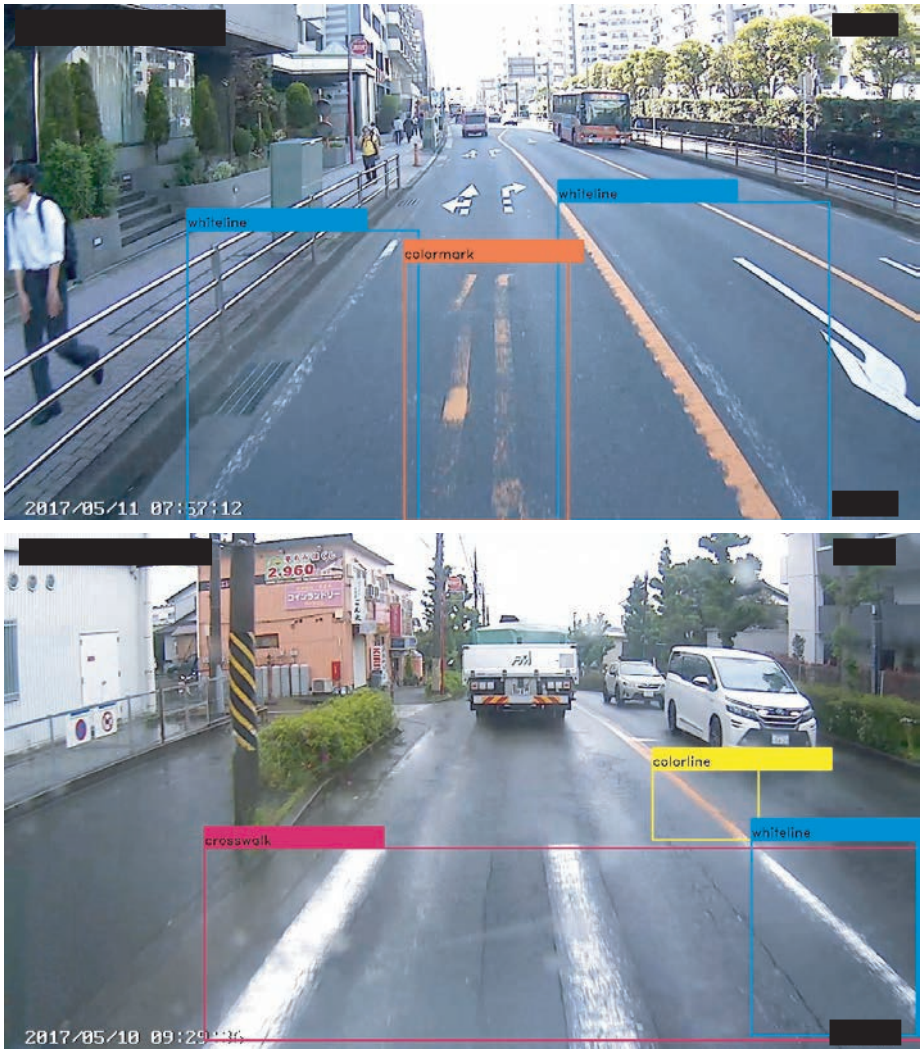


Figure 5.6: Examples of our dataset, RoadDamageVOC. The blurred road markings are surrounded with bounding boxes.

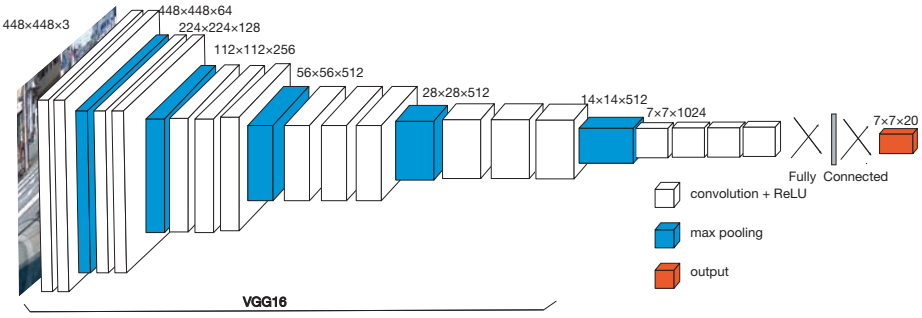


Figure 5.7: The model architecture of which we use in our experiment. The combination of VGG-16 [72] and YOLO [101] model.

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{blur}\left(C_i-\hat{C}_i\right)^2$$

$$+\lambda_{noblur}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noblur}\left(C_i-\hat{C}_i\right)^2$$

$$+\sum_{i=0}^{S^2}\mathbb{1}_{ij}^{blur}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2$$

where $\mathbb{1}_{i}^{blur}$ denotes if road marking blur appears in cell i and $\mathbb{1}_{ij}^{blur}$ denotes that the jth bounding box predictor in cell i is required to predict. The first two terms are for penalizing bounding box position and size errors. The third term represents confidence prediction error; boxes that do contain objects should have a high C while those that do not should have a lower C. The last term penalizes object classification error. To stabilize learning, we balance the loss function with some coefficients that control the influence of each term towards the total loss.

The model architecture is constructed with two parts of neural networks and is depicted in Figure 5.7. This architecture is based on original YOLO model [101]. The first part of neural networks is feature extractor. The architecture in original YOLO called darknet, which is pre-trained with ImageNet [54]. However, it is difficult to reuse the pre-trained darknet weights because of their implementation. Therefore, we use pre-trained VGG-16 [72] architecture and its weights. The second part of neural networks is as same as original YOLO except for the number of classes C.

5.4.3 Single Shot Multibox Detector

The second method to detect the damaged road markings is single shot multibox detector (SSD) [102]. Given the large objects in images, SSD has been reported that it is superior to detect the big object. Since road markings often appear in the images largly, we assume that SSD would be suitable for our task.

SSD is built up with multiple convolutional layers and pooling layers which reduce the size of feature maps. In more detail, SSD consists of a VGG16 network [72] and six convolutional layers. SSD outputs the bounding box canditates as a result by using the output of VGG16 and the six convolutional layers. One bounding box is represented by a location vector $l = (x, y, w, h)$ and the class vector $p \in \mathbb{R}^C$. Note that x and y represent the center coordinates of the bounding box, and w and h represent the width and height of the bounding box respectivly. Simultaneously, p is a C-dimentional vector and C is the number of classes to be classified. Namely, each element p_c of p vector is the probability of the class which the bounding box belongs to; $\sum_c p_c = 1$. In the case of our task, the number of classes C becomes the number of kinds of road markings which are to be detected. SSD only outputs the bounding box whose highest probability p_c exceed the threshold λ. Therefore, by adjusting the threshold λ, we can control the number of SSD outputs, the bounding boxes. In the experiment, we validated how the accuracy change when the threshold λ change. The objective function of SSD is as follows:

$$\mathcal{L}(m, c, l, g) = \frac{1}{N}(\mathcal{L}_{conf}(m, c) + \alpha\mathcal{L}_{loc}(m, l, g)), \tag{5.2}$$

where the $\mathcal{L}_{conf}$ and $\mathcal{L}_{loc}$ represent the cross-entropy loss and mean-squared loss respectivly. Moreover, $m^p_{i,j} = \{1, 0\}$ becomes 1 when the ith bounding box and the jth bounding box that the class of these boxes indicates are c, are coincided, or 0 when it is not. The boxes being coincided means that the coefficient of Jaccard is more than 0.5. Note that α is the hyper-paremeter to adjust the balance of two loss function and we follow the manner of the original paper: $\alpha = 0.5$.

5.5 Implementation

5.5.1 Hardware Side

For the environment to recognize and/or detect the road marking damage, we adopt the NVIDIA Jetson TX2, which is operated on Linux based OS (i.e Ubuntu). Otherwise, there is a RaspberryPi that is also operated on Linux based OS as well. However,

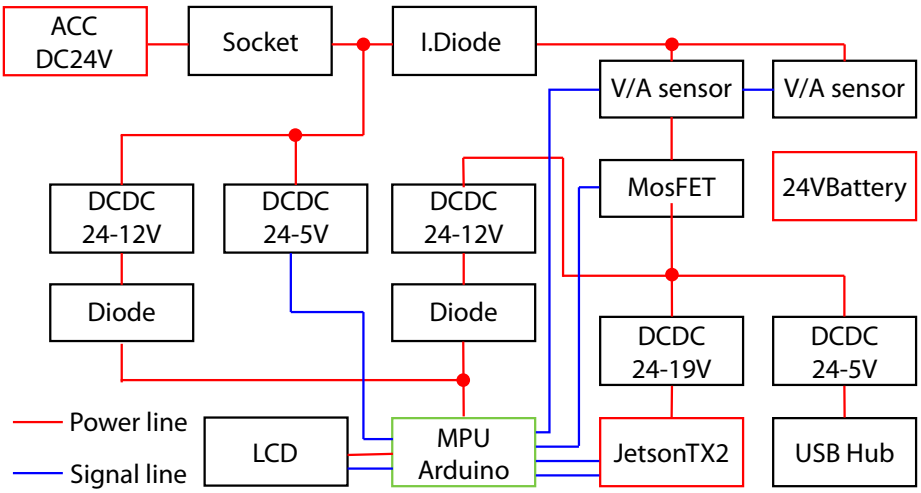


Figure 5.8: System Configuration of the hardware part of CityInspector. The red line represents electric currents and the blue line represents the signal line.

utilizing the RaspberryPi, the road damage recognition and/or detection is taken on CPU. On the other hand, the Jetson TX2 can conduct the road damage recognition and/or detection on their GPU, which is installed into the JetsonTX2. Of course, while the price of the Jetson TX2 is much higher than that of the RaspberryPi, we prioritize the performance of road damage inspection and we use JetsonTX2.

The system configuration of the hardware part of our CityInspector is depicted in Figure 5.8. The 24V voltage is applied on the cigarette lighter port in a garbage truck when the engine of it is started. We use Arduino to monitor whether the voltage is being applied or not. While the voltage is being applied on the cigarette lighter port, an electric current flows into the JetsonTX2 and a built-in battery to charge it. We call this status as *active*. When the engine of the garbage truck is stopped, Arduino detect it and start the countdown timer and the electric current flows from the built-in battery instead. This countdown timer is used for a temporal engine stop that the city officers take a rest. This status is called *countdown*. If the engine is restarted during the countdown, the status change from *countdown* to *active*. If not, the status change from *countdown* to *inactive*, which is that the JetsonTX2 get a shutdown signal from Arduino. CityInspector provide us a LCD monitor to confirm the status as shown in Figure 5.9. The completed prototype of CityInspector is depicted in Figure 5.10.

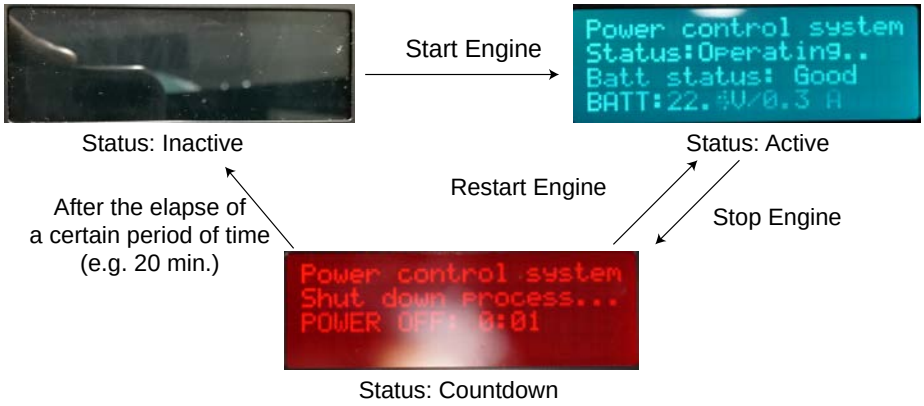


Figure 5.9: The state transition diagram with the actual monitor of CityInspector. There are three status.

5.5.2 Software Side

Regarding the software side, there are several modules constructing CityInspector as shown in Figure 5.11. First, the main module, that is road damage detection, gets the images from an equipped USB camera and processes them to detect the road damage. At the same time, the geolocation acquiring module outputs the geolocation: longitude and latitude. Then the detection result sending module aggregates the results and send them. At the visualization side, the detection result receives module receives the results from N of CityInspector that garbage trucks equip. Lastly, the result filtering module sort the results so that each organization using CityInspector can see the status of road damage at their jurisdiction, respectively as shown in Figure 5.12. Figure 5.12(a) is a screenshot of the visualization tool. We can see the number of the place where the road is damaged by the number and colors. Simultaneously, Figure 5.12(b) shows the detail, with each marker represents the kind of road damage markings and enables to check the status of road damage by an actual image.

5.6 Experiment

In this section, we show two kinds of experiments. First, we compare DoE with baselines which are used in previous works to evaluate DoE. Then, we examine whether DoE is fit for practical use.

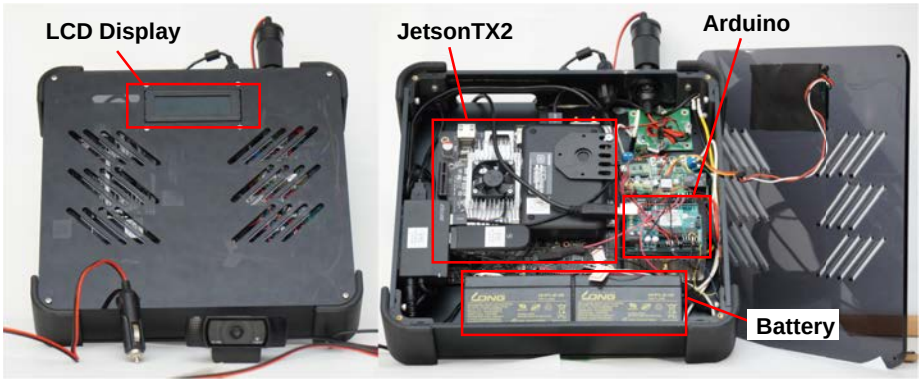


Figure 5.10: Prototype of CityInspector. The left image shows the outside appearance of CityInspector and the right image shows the inside structure. CityInspector contains NVIDIA JetsonTX2, Arduino and built-in battery.

5.6.1 The Accuracy of Road Recognition

In order to evaluate DoE and its architecture, we compare it with previous work [97][98]. Although Maeda et al. [97] classify the degree of road condition into three types: "smooth (no-damaged)", "need repair" and "not need repair", its actual classifications are difficult to distinguish, as different outputs are produced from visually similar images. To make this problem more interpretable, we simplify this task as binary classification problem: the road which is photographed in given images is whether damaged or not. Therefore, we used the dataset we use consists of images labeled "damaged" and "undamaged" in Table 5.2.

As baselines, we adopt two kinds of methods. The first method tests classic machine learning algorithms: a support vector machine classifier (SVM) that can achieve good performance at binary classification, and a random forest which can detect the line damage [98] as well as our work. The second method is a deep neural network. We choose the AlexNet which is proposed in [46] and won the ILSVR competition in 2012, and has been used quite actively since [97]. Further, since the aim of our study is road detection on an edge computer, the smaller model is desirable and we also examine the alternative models that are proposed in [97]: AlexNet–(d) and AlexNet–(e).

Before training DoE, we initialize the weights of DoE with random values and use the Adam[106] stochastic gradient descent algorithm with a learning rate of 0.0005, a momentum of 0.9 and a batch size of 32. Meanwhile, those of baseline networks use respective values of $1e^{-4}$, 0.9 and 100. We then trained models with early stopping,

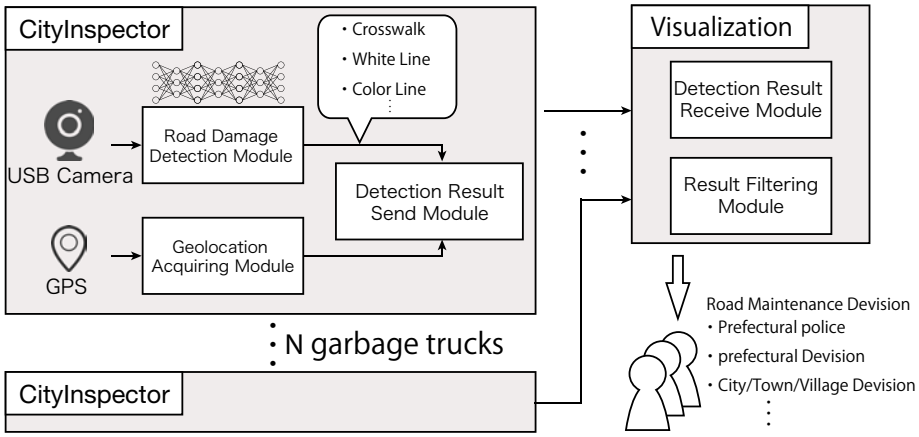


Figure 5.11: Software System Configure.

Table 5.4: Accuracy Comparison on the Line Damage Binary Classification Task.

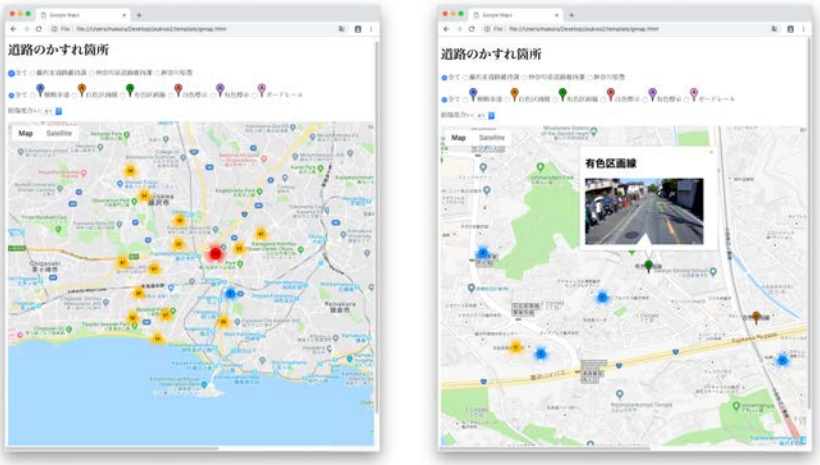
Method	Acc.	AUC	Recall	Pre.	F1	Params.
Linear SVM	82.4	0.82	0.87	0.83	0.85	–
Random Forest [98]	84.0	0.83	0.91	0.83	0.87	–
AlexNet [46]	92.5	0.9833	0.92	0.92	0.92	58000K
AlexNet–(d) [97]	92.5	0.9845	0.92	0.92	0.92	1680K
AlexNet–(e) [97]	92.7	0.9859	0.93	0.93	0.93	913K
DoE (ours)	**94.1**	**0.9894**	**0.94**	**0.94**	**0.94**	**18K**

which is a training procedure that stops training if the error on the validation set stops decreasing.

Table 5.4 shows the experiment result. DoE outperformed baseline methods, even though the number of parameters is quite less than others. This result shows that deep architectures do not necessarily have good performance in computer vision tasks, even if it has been reported as good architecture. In short, it is necessary to tweak model architectures for specific tasks.

Practice Investigation

For practical use, we examine whether DoE is able to detect line damage from an actual image from a camera. For this, we retrain DoE from a binary classification problem to a 3-class classification problem; “undamaged (no-damaged)”, “damaged”, “no line”. When we use DoE that solves a binary classification problem, it may cause false de-



(a) Overall. (b) Detail.

Figure 5.12: Visualization of the road damage inspection. (a) The visualization tool gives a global overview of the number of the place where the road is damaged by the number and colors, (b) each marker represents the kind of road damage markings and enables to check the status of road damage by an actual image.

tection when there is no line. The result confusion matrix of 3-class classification is shown in Table 5.5 and actual detection in Figure 5.13. As a result, DoE can classify the road condition with 98% accuracy. Furthermore, in the case of Figure 5.13(a) (b), DoE classifies the patches perfectly. Note that in (b), at the location of the yellow font, the left upper patch is classified as “undamaged” correctly, while patches right side hand of it is classfifed as “damaged”. On the other hand, DoE misclassifies “undamaged” patches as “damaged”. This might happen if the line is dirty or something is on the line (e.g. the shoe is photographed in Figure 5.13(d)).

5.6.2 Road Damage Detection Accuracy

In order to evaluate whether our models is able to detect the road markings blur, we calculate the mAP score. The mAP calculation is conducted as follows:

1. Calculate the average precision (AP) for each class based on model prediction. AP is related to the area under the precision-recall curve for a class. If the IOU (illustrated in Figure 5.14) exceed T_{mAP}, we consider the prediction is a correct detection.

Table 5.5: Confusion Matrix of Deep on Edge Model.

Number of Parameters 18171		Prediction			Recall
		Undamaged	Damaged	No line	
Ground Truth	Undamaged	2795	162	0	0.945
	Damaged	196	3734	2	0.950
	No line	0	1	2012	1.00
Precision		0.945	0.950	0.999	0.980

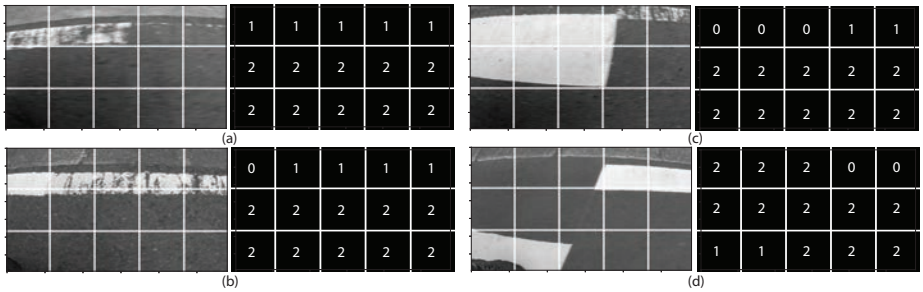


Figure 5.13: The result of the line damage detection with actual images. Each number denotes the classes, respectively; 0: undamaged, 1: damaged, 2: no line. (a) (b) DoE classifies all patches correctly. (c) Although DoE mistakes classifying "undamaged" as "no line" (at red fonts), it correctly detects damage at yellow fonts.

2. Take the mean of these average each class-precision.

YOLO model

In our experiment, we change the threshold T_{mAP} and T_{NMS} that is used for calculating mAP score and executing non-maximum suppression (NMS) algorithm respectively, in order to observe the mAP score change.

We adopt Adam optimizer [79] and our learning rate schedule is as follows: For the first epochs we set the learning rate 10^{-4}. If we start at a high learning rate (e.g. 10^{-3}) the gradient explosion often occurs. We continue training after first epochs for 50 epochs, then 10^{-4} for 30 epochs, and finally 10^{-5} for 20 epochs. Note that the batch size is 16.

The result of our experiment is shown in Table 5.6. Although the mAP is not so high, the mAP score increases as the T_{mAP} decreasing. This means that the main error of prediction is incorrect localization. Furthermore, while the AP score of white lines is

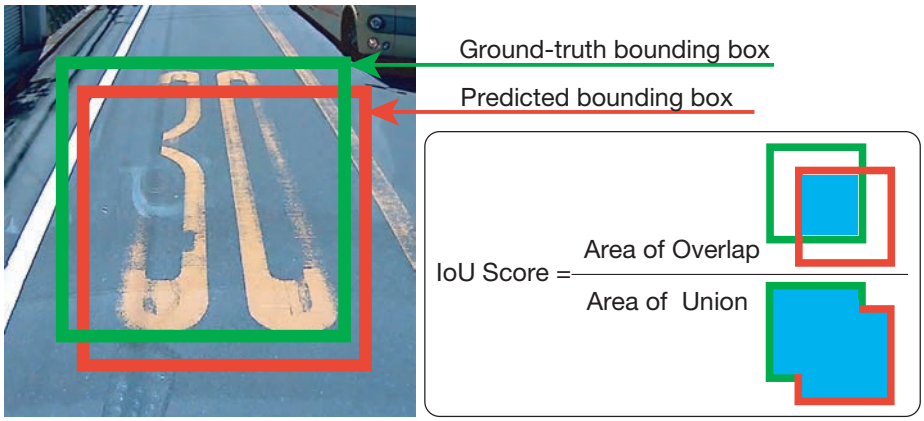


Figure 5.14: The illstration of intersection over union.

Table 5.6: The Results by Our YOLO Model.

T_{NMS}	T_{mAP}	mAP	White line	White mark	Color line	Color mark	Crosswalk	Guardrail
0.4	0.5	14.9	51.7	10.0	0.1	0.0	22.4	5.4
	0.3	21.5	56.1	20.5	3.2	0.0	33.4	15.9
	0.1	29.1	61.8	27.4	4.7	0.0	37.7	42.7
	0.0	30.9	65.4	28.4	5.7	0.0	38.7	46.8
0.9	0.5	16.0	53.1	9.1	0.1	0.0	26.0	7.1
	0.3	22.3	57.6	22.4	2.8	0.0	34.4	16.5
	0.1	29.3	61.9	29.1	4.3	0.0	38.2	42.1
	0.0	30.7	65.4	29.8	5.2	0.0	39.0	44.5

high, that of color marks is all zeros. We think that this is due to the number of ground truth of color mark is quite few. Meanwhile, comparing the mAP score at $T_{NMS} = 0.4$ and at $T_{NMS} = 0.9$, the higher the threshold is, the higher the mAP score is. T_{NMS} control the number of bounding boxes of prediction by suppressing the bounding boxes which are overlapping each other over T_{NMS}. Although there is a risk that mAP score decrease due to excess bounding box, increasing the number of boxes of prediction is effective.

SSD Model

Table 5.7 shows the result of the accuracy of our SSD model when the class threshold is changed. The class threshold is changed with $\lambda \in \{5.0 \times 10^{-3}, 1.0 \times 10^{-2}, 5.0 \times$

Table 5.7: The Result by Our SSD Model.

Class Threshold	mAP	White Line	White Mark	Color Line	Color Mark	Crosswalk	Guardrail
5.0×10^{-3}	48.80	49.12	51.71	33.77	51.31	63.04	43.85
1.0×10^{-2}	48.65	49.07	51.42	33.77	51.17	62.63	43.85
5.0×10^{-2}	46.95	48.57	49.74	32.33	50.50	57.16	43.42
1.0×10^{-1}	45.75	46.39	49.67	32.33	46.51	57.16	42.42
5.0×10^{-1}	39.13	41.02	40.31	29.02	34.87	50.68	38.88

Table 5.8: The Time Result by Our SSD Model

Class Threshold	Time
5.0×10^{-3}	43.63 ms ± 220.72 μs
1.0×10^{-2}	30.75 ms ± 44.82 μs
5.0×10^{-2}	23.97 ms ± 65.55 μs
1.0×10^{-1}	23.31 ms ± 87.78 μs
5.0×10^{-1}	22.68 ms ± 72.84 μs

$10^{-2}, 1.0 \times 10^{-1}, 5.0 \times 10^{-1}\}$. The mAP score increase as the λ is set as low. This is because the number of the bounding box that is the output of SSD becomes large when the λ is high. With this the bounding box increasing, the number of box that matches to the ground truth bounding box becomes large. When we prioritize the performance of CityInspector, that is the accuracy of road damage inspection, the class threshold should be set as low value. On the other hand, it takes more times to output due to the number of the bounding box. Hence, the time for the road damage inspection and the accuracy is trade-off. We discuss this trade-off in later.

Then, we validated the accuracy change when we change the size of the training data. We randomly sampled the subset of training data with the rate with in 0.1 to 1.0. The result of that the SSD was trained with each training subset data is shown in Figure 5.15. As the illustrated, the accuracy becomes higher as the size of dataset increase. In more detail, when the rate is more than 0.4 (i.e the size of training data is $0.4 \times 1807 = 722$), the accuracy increase slowly. From this observation, we can expect that if we gather the training data at least 700 when we apply the CityInspector to new cities, it would perform as well as Fujisawa city. Of course, we can also expect that the more training data, the higher the accuracy becomes.

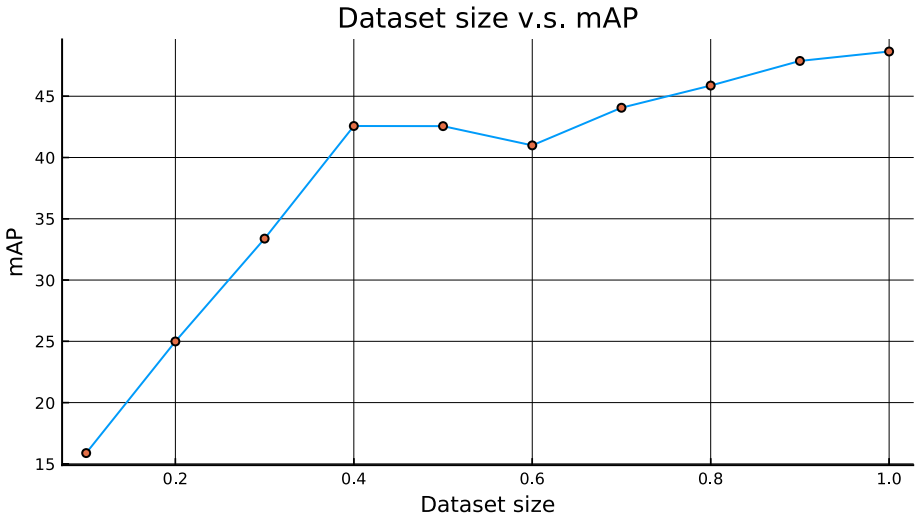


Figure 5.15: The mAP score changes as the size of the training data increases.

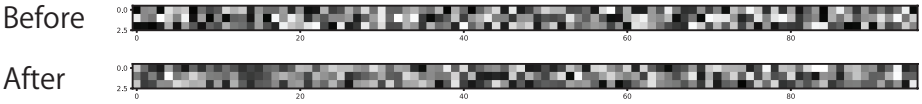


Figure 5.16: The visualization of kernels of first convolution layer of DoE before training and after.

5.7 Discussion

5.7.1 Activation Visualization

In general, while it is said that CNNs are useful for image recognition, it is difficult to understand what the network learn. For instance, Figure 5.16 is the visualization of the kernel of first convolution layer of DoE before training and after. To tackle this problem, we visualize the activation of each kernel of DoE when the input damaged road image comes as shown in Figure 5.17. From the visualization, we can see that the model activates the part of damaged line like noisy dots, while there are only a few activated on the undamaged line. Remarkably, at the fifth layer, the activation of each unit in each image is mostly opposite. This result shows that DoE correctly learns the damage of line.

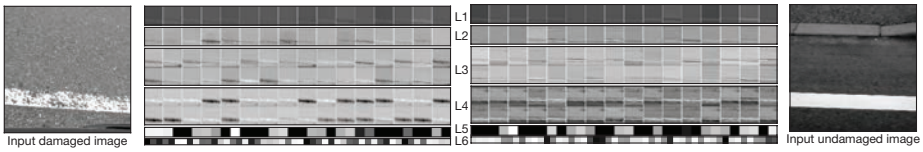


Figure 5.17: The visualization of the activation of each layer when the damaged line and undamaged line input images come. Remarkably at the fifth layer, each activation is active oppositely.

5.7.2 Input Image Generation

Furthermore, to understand the model in detail, we generate the image that DoE is likely to classify to damaged and undamaged. This method is inspired by [107]. The output of DoE is through a sigmoid function which has the asymptote $y = 0$ and $y = 1$ and the nature:

$$\lim_{x \to \infty} sigmoid(x) = 1 \tag{5.3}$$

$$\lim_{x \to -\infty} sigmoid(x) = 0. \tag{5.4}$$

Therefore, sigmoid is likely to output nearly 1 when it receives a large input and vice versa. Utilizing this nature, we can observe the output of DoE changes with fixed model parameters when we change the input. Beginning with a randomly initialized image, we use gradient ascent:

$$x \leftarrow x + \eta \frac{\partial a_i(x)}{\partial x} \tag{5.5}$$

to change an input image. Note that x denotes generated image input and η denotes learning rate. Furthermore, $a_i(x)$ represents the output of the ith layer and we use the last layer$i = 7$. Then, we maximize and minimize the output of $a_7(x)$ by Eq. 5.5. To emphasize the features which model learned, we applied Lp norm regularization:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}} \tag{5.6}$$

and total variation:

$$V(x) = \sum_{i,j} \sqrt{|x_{i+1,j} - x_{i,j}|^2 + |x_{i,j+1} - x_{i,j}|^2}, \tag{5.7}$$

which smoothed the images. The results are depicted in Figure 5.18. The image in the left of Figure 5.18 is classified by DoE as “undamaged”, and the right image is

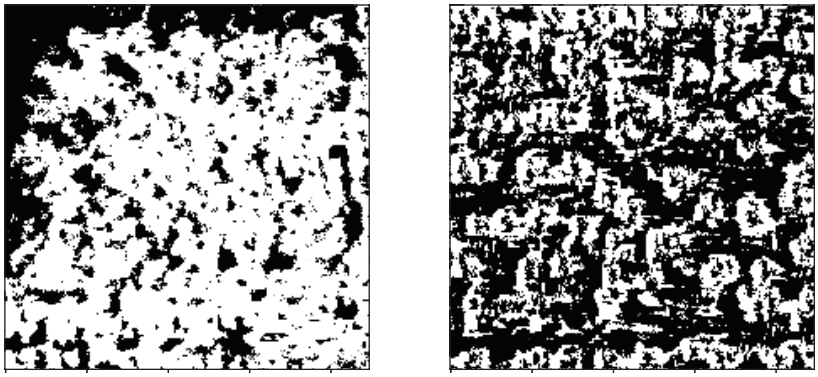


Figure 5.18: The examples that are generated to let DoE more likely to classify as “undamaged”, and vice versa. There are more the white parts in “undamaged” image than “damaged” one.

classified as “damaged”. There are much more white parts in the “undamaged” image than the “damaged” image. This shows that DoE recognizes line damage. In summary, from these visualizations, we found that DoE has learned to extract patterns and differentiate “damage” and “no damage” from the dataset, without any clues except from given labels.

5.7.3 The performance of Road Damage Detection

YOLO Model

Overall, the mAP and AP score of each class is not high. To understand this result, we visualize the prediction on the actual image with ground truth. The visualization is shown in Figure 5.19. As a Figure 5.19 (a) (e) and (f), the correctness and incorrectness of prediction is very obvious. On the other hand, in Figure 5.19 (c) and (d), while the labels of prediction match to those of ground truth, the IOU of prediction is very low. This cause the mAP score being lower. When the threshold is very low, this prediction becomes correct. One of the ways to prevent this mAP decline is surrounding the blurred road markings with bounding box more exactly.

The very interesting point is depicted in Figure 5.19 (b). In this image, the prediction is correct at guardrails and white line and incorrect at color line. However, there is a prediction of white line and it seems there is a blurred white line, despite of no an-

Table 5.9: Inference Time at the Each Device

Device	Time
Intel Core i7 2.9 GHz	$848ms \pm 13.2ms$
NVIDIA TitanX（Pascal）	$9.59ms \pm 14.3\mu s$
NVIDIA 1080Ti	$12.8ms \pm 5.67\mu s$
NVIDIA JetsonTX2（CityInspector）	$188ms \pm 141\mu s$

notation of ground truth. This might be happened because of the annotation definition inconsistency.

In summary of these analysis, the more accurate the annotation is, the higher the mAP score becomes.

SSD Model

In the experiment, we use NVIDIA TitanX (Pascal) graphics cards to detect the road damage from the RoadDamageVOC. However, we are planning to use JetsonTX2 for CityInspector, so that the actual inference time differs from Table 5.7. For the comparison between the devices, we measured the time which each device takes to inspect one frame image as shown in Table 5.9. While the time of JetsonTX2 is spent much longer than that of TitanX, is much shorter than that of CPUs. Although the time of JetsonTX2 is $188ms$, it is still within the required time.

Moreover, we investigated the output of our model and the result of outputs changing the class threshold is depicted in Figure 5.20. As we mentioned in approach section, the number of bounding box outputed increase when the class threshold λ is low (i.e $\lambda = 5.0 \times 10^{-3}$). On the other hand, the number of bounding box becomes small when the class threshold λ is high (i.e $\lambda = 5.0 \times 10^{-1}$). Given the actual application, the proper value of the class threshold would be set as high. This is because the garbage trucks run the same road again and again during their work and the road markings are damaged not so frequently. For this reason, the CityInspector have to detect the sure damage even if it makes mistakes a lot. When the same road is detected as being damaged in many times, the confidence of the road damage will be much higher. In short, the proper value of the class threshold should be set depending on the frequency of that the garbage trucks run on the road.

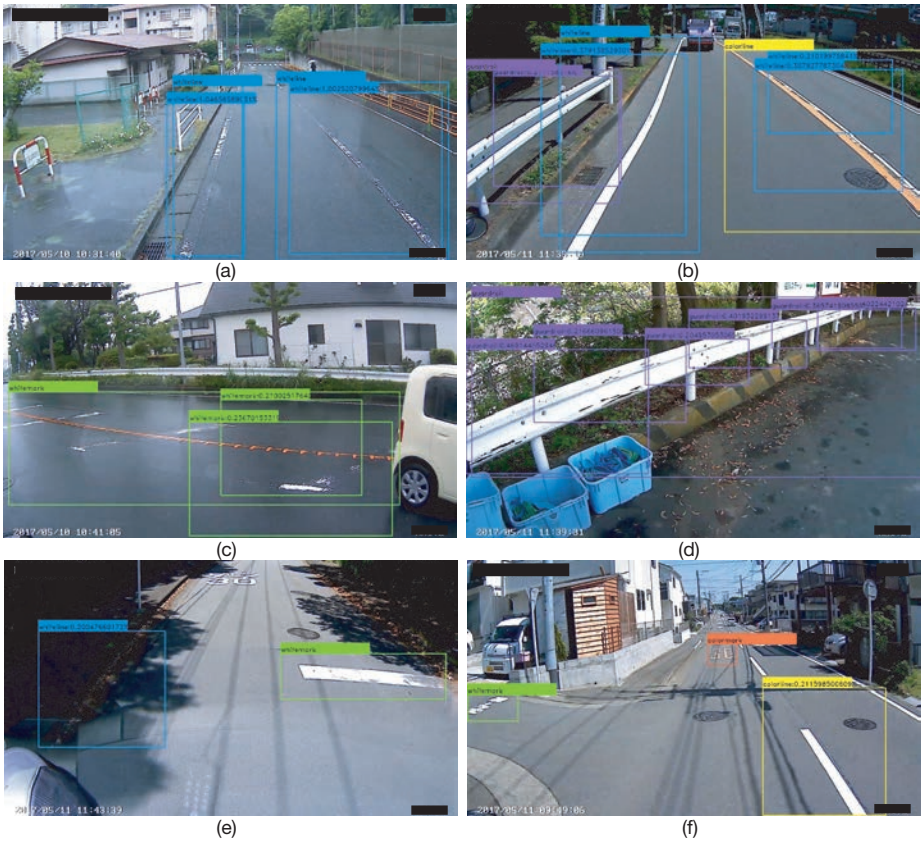


Figure 5.19: The result of model prediction. The bounding boxes whose label contains the confidence are the prediction and the boxes without confidence are the ground truth. (a) The prediction is perfectly correct. (b) The prediction of white line and guardrail is correct and the color line prediction is wrong. However, the model detects the white line blur. (c) (d) The label of each prediction is correct (white mark and guardrail respectively), but the IOU does not exceed threshold, thus these prediction is calculated as False.

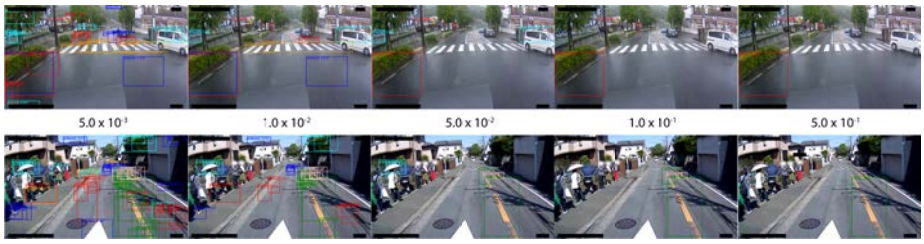


Figure 5.20: The result of output that the number of the bounding box changes due to the class threshold λ changing.

Economic Cost Reduction

With deploying our CityInspector to the city, the current work that city officers patrol the city to inspect the road condition in 3 times in a week can be replaced so that the cost about 0.6 man-month is reduced in theory and instead the workers can do other jobs. Moreover, conducting road inspection not sending and processing images in the cloud computers but processing in the edge device such as our CityInspector can reduce the communication costs which are simulated as follows:

1. The videos are filmed during the garbage trucks working: 8:00–15:00 (300 min.) and 1 GB with low-resolution.
2. About 50 garbage trucks runs so that 50 GB of videos are filmed in a day.
3. They work every day and every week. Therefore, the total size of video becomes 1200 GB in a month.
4. The cheapest SIM in Japan is 5K yen per 30GB.

Consequently, the total cost about 200k yen would be reduced.

Simultaneously, by installing CityInspector into garbage trucks, city officers who usually inspect the road condition in daily do not need to conduct the inspection and they can work other jobs. For example, in Kanagawa Prefecture, the city officers go outside to inspect the road condition in three times a week. Thus, if that works are replaced by CityInspector, the time which has been spent for the road inspection can be used for other works; about 0.6 man-month is reduced.

5.8 Summary

We presented a system which detects the road marking blur via object detection approach, which is based on deep convolutional neural networks. We assumed our system is deployed on garbage trucks and works with the images from the drive recorder of garbage trucks. To train our model, we develop a new dataset that contains the images from drive recorders with the annotation with respect to blur; white line, white mark, color line, color mark, crosswalk, guardrails. Our experiments show that object detection approach performs well at road marking blur detection. In summary, CityInspector is the first trial to assume edge devices are widely spread in the city and to realize the varous cameras to be able to capture the context of city; detecting road damage by using the dashboard cameras.

Chapter 6

CityFlow: Urban Integrated Development Environment

6.1 Introduction

In order to realize *smart cities*, there has been extensive research into using both machine learning (ML) and deep learning (DL) techniques. Note that we omit indicating ML and DL as just ML from this. For example, citizens' behaviour prediction by using GPS trajectories of their smart devices or taxis [13], and road damage detection from video recorders mounted on automobiles [108]. In both cases, city data is gathered into centralized cloud servers for analysis. This centralized approach is well suited to ML algorithms, but unfortunately does not reflect the reality of Smart Cities. Typically, cities have a multitude of sensors and 'things' (e.g. automobiles, mobile devices and robots) that are distributed throughout the city and communicate with each other via the Internet (e.g. Wi-Fi, LTE, 3G or Ethernet). This results in city data exhibiting a wide variation in time and space both in short term, i.e. daily cycles, and in longer term cycles as the city, its infrastructure and its citizens evolve. In short, the city is a spatial-temporal distributed edge environment (STDEE) and requires we adapt our ML techniques to this environment. However, developing and operating ML applications in a STDEE is not straightforward. This is primarily because the development process, which is already complicated because of the need for multiple trials to aid learning, needs to also accommodate the distributed nature of the STDEE. Obviously this leads to significant development costs, both in terms of time and resources, and is by nature inherently complex.

To ease both the cost and complexity, an integrated development environment is desired for ML application development suitable for STDEEs as found in smart cities. To solve this problem, we have designed and implemented *CityFlow*, which supports the development of ML applications in the STDEE city. CityFlow makes it possible to easily describe the flow of data which comes from devices in the city, and to deploy trained ML models to a variety of devices distributed throughout the city. By making these processes easy, we can conduct data preprocessing and preservation quickly and repeat *proof of concept* (PoC) that verify the performance of the ML model. CityFlow is built using Distributed Node-RED (DNR) [109, 110]. Node-RED is a visual tool for the IoT, interconnecting hardware devices, Web service APIs and online services quickly. DNR is an extension of Node-RED that supports an edge (or fog) computing environment.

This thesis reports two case studies we conducted to evaluate the usefulness of CityFlow. The first study develops a road damage detection application which identifies

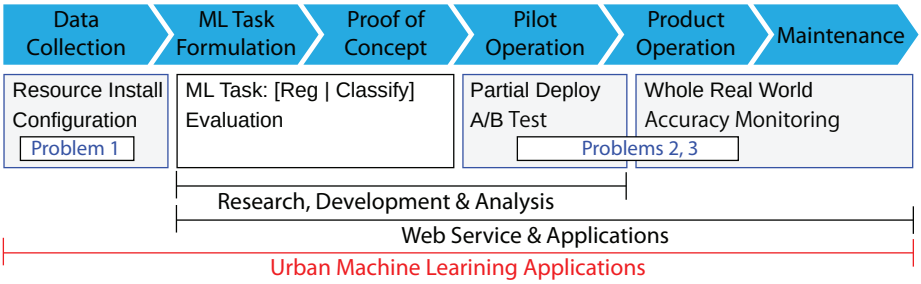


Figure 6.1: General life cycle of machine learning applications. There are several types of application to serve their purpose: Research, Web services, urban application.

damaged road and road markings from video images gathered by city garbage trucks. This application utilizes an object detection method via deep neural networks and also protects privacy in visualization. The second study develops an automatic labeling application that allows city officials to report problems which they find in the city, such as the illegal dumping, uncollected garbage, graffiti, etc. Images attached to the reports are classified automatically into the type of problem making reporting easier and less error prone for city officials. The contributions of this section are summarized as follows:

- We identify key issues in the development and operation of ML applications in spatial-temporal distributed edge environments, such as smart cities.
- We design and implement CityFlow, the integrated development environment to address those issues.
- We validate our approach by reporting on three case studies that we explain in previous chapter and additional case of participatory sensing we have designed and deployed to aid city staff.

6.2 Urban Machine Learning Applications

6.2.1 Spatial-Temporal Distributed Edge Environments

With the developments in IoT technologies, an enormous amount of data from sensors in the city can be obtained. This covers traditional city infrastructure such as water, electricity and lighting but also increasingly comes from smart devices such as smart

transportation, smart homes and offices and perhaps most importantly citizen data from their personal devices. At the same time, an increasing number of small edge devices such as embedded computers being deployed: such as NVIDIA Jetson TX2 (embedded computers), Google Embedded TPU and Xilinx Spartan6 (FPGA).

As the capabilities of the these edge devices increases, regardless of their low price, it is possible to use them to support distributed, ML applications. For example, they are already used in domains such as agriculture[1]. Simultaneously, some work has begun to explore deep learning on edge computers [111] for mobile applications. Finally, an important trend is towards treating city information from web service [2] and also the statistics of the city from open data thanks to the spread of IT technology, such as cloud computing technology, which treats the data sources as virtual sensors.

While these trends are leading to significantly more city data being gathered, it is also clear that the spatial-temporal nature of that data is changing. Significant amounts of data are spatial in nature, e.g. where sensors are mounted on garbage trucks, the data is gathered from a variety of locations as the truck travels through the city. Equally, data is gathered during the working day, but not at night - hence it exhibits a high degree of temporal variance. This variety is common to many sources of data, for example data from citizens' mobile devices.Consequently, cities where the IoT and the edge computing technologies are spread throughout exhibit key characteristics of STDEEs.

6.2.2 Life Cycle of Machine Learning Applications

In general, the phases (or life cycle) which are conducted in ML application development are shown in Figure 6.1. In the first phase, in order to tackle a particular city problem, data analysis such as statistical estimation is carried out for understanding the gist of the problem and formulate it as a ML task. Most of the task formulation becomes either regression tasks or classification tasks. Simultaneously, the ML approach would be determined by whether the data is labeled (or annotated) fully, partly, or not at all: supervised learning, semi-supervised learning, unsupervised learning.

In the second phase, to perform the formulated ML task, we design and train the ML model using a given dataset. Then, we experiment to evaluate the model performance as regards its accuracy. In this thesis, we refer to this processes as the Proof of Concept (PoC). If the model performance is poor, it may be necessary to start over by

[1] https://cloud.google.com/blog/products/gcp/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow

[2] http://soramame.taiki.go.jp/

re-formulating the problem.

As a result of the PoC, if the ML model is able to perform adequately, then we implement and install this ML model into the edge environment and verify whether the model is effective in the real world (we refer to this as the pilot operation). During this phase we adjust for accuracy; identifying factors such as data quality or quantity, which can be improved or factors such as the capacity of the model which may result in re-design of the model.

Finally, we shift to the product operation phase during which, in order to improve the model, we periodically re-build the dataset with additional new data which continues through to the maintenance phase.

In summary, in the development and operation of the ML application, we repeat the exploratory phases {task formulation, PoC, pilot deployment, production operation, and maintenance}.

6.2.3 The Problems for Machine Learning Applications in the City

While urban ML applications, which are offered to solve the urban problems (e.g. parking availability monitoring [112]) are proposed, there is some difficulty to deploy to the real world. The data used by those applications are collected by sensors which are installed temporarily, with limited coverage, and in a precise controlled manner. Furthermore, they have to select the sensor devices which satisfy their application requirements. In this thesis, we call these technical concerns *spatial-temporal device-data dependency*, which occurs in the phase surrounded by blue line in Figure 6.1.

Problem 1: Urban Data Collection and Preprocessing.

In statistical machine learning, it is generally assumed that the data have already been collected, formatted and normalized to use for the training dataset - often using offline processing and assuming homogenous datasets. In contrast, cities exhibit highly heterogeneous datasets, which change over time as the city changes and where data is streamed in real-time. This results in increased processing and storage and requires a dynamic learning model that retrains as data changes. While there exist a number of algorithms which are able to handle streaming data, such as CQL [113], Apache Spark [114] and Storm [115] they often assume cloud based servers with load balancing and scaling, and work on homogenous datasets. Which, as discussed, is not often the case with real world city datasets.

In addition, we usually conduct some pre-(post-)processing of the data. For example, for privacy protection, since the data from edge devices is often bound to the real world, there is a risk of privacy invasion. For example, camera data from city sensors, or car data captured in real time has significant privacy issues. However, the diversity of devices makes it hard to cope with the data processing required for dealing with this privacy invasion.

Problem 2: The Machine Learning Model Execution Environment.

After the PoC phase, it is necessary to distribute the trained ML model to all (or a subset) of the edge devices in the city. In typical ML applications, high-performance cloud computers are utilized. This is often necessary because the ML model has a large number of parameters requiring significant memory and high-performance CPUs. However, it is often too expensive for cities to own and manage such servers due to budget constraints. In contrast, we can use edge devices in a STDEE city that typically have significantly lower specifications than cloud based servers. However, while we can use traditional approaches on edge devices with reasonable computational resources, the latest approaches which have good performance, such as deep neural networks, are not always appropriate due to their resource needs. In order to benefit from their performance, we need to divide the ML model into subsets of small ML model or processes that can be distributed across a set of edge devices [116, 117] or compress it to load it on the edge device memory [111].

Additionally, since the edge devices are highly heterogeneous, it is not just a simple matter of distributing partial modes to a set of edge devices, rather the constraints and context of each device needs to be considered as part of any distribution algorithm.

Problem 3: Covariate Shift of the City Data.

In statistical machine learning, typically it is required that there is no difference between the marginal probability distribution of the training dataset and that of the test dataset. Namely, assuming the distribution of training dataset is p and that of test dataset is p', $p(\mathbf{x}) = p'(\mathbf{x})$ is required. In an STDEE based city, the network status and the context of the location where the edge devices are installed, changes. Therefore, the marginal distribution becomes different, although the relationship between the data $\mathbf{x}$ and the desired output y, that is posterior $p(y|\mathbf{x})$ is consistent. This phenomenon is known as *covariate shift* [118, 119]: $p(\mathbf{x}) \neq p'(\mathbf{x}), p(y|\mathbf{x}) = p'(y|\mathbf{x})$. When considering

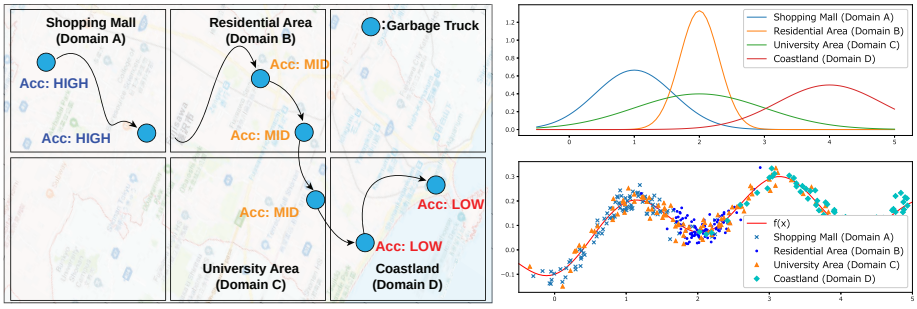


Figure 6.2: Influence on accuracy due to a spatial-temporal covariate shift. For models taught from data around commercial facilities, prediction accuracy is high in similar commercial facilities. The prediction accuracy decreases as the spatial features differ.

smart cities, we use the hypothesis that the covariate shift exists in the city, as the distribution of the data from the city often varies significantly in spatial and in temporal domains. We call the covariate shift caused in the city as Spatial-Temporal covariate shift.

The covariate shift of the city data is illustrated in Figure 6.2. The prediction accuracy of the model is high in the shopping street, because the model is trained with the data collected in the shopping street. In contrast, in other areas away from the shopping centre, the more the data distribution differs from that of the shopping street, the lower the prediction accuracy is owing to the ST covariate shift. For instance, supposing car detection based on video image analysis is required, a model which is trained with the video of shopping streets can detect cars with high accuracy in similar streets. By contrast, at the sea coast, the model struggles to precisely detect cars because the background is sea water instead of buildings. Similarly, if the model is trained with the videos taken in daytime, it is difficult for it to detect cars during the night, even though the location is the same. Therefore, it is important to consider these ST covariate shifts when we develop the ML applications for the smart city.

6.3 CityFlow

To remedy the spatial-temporal device-data dependency, we propose CityFlow, which is the system combining *Distributed Node-RED* and *Sensor over XMPP*. The overview is depicted in Figure 6.3.

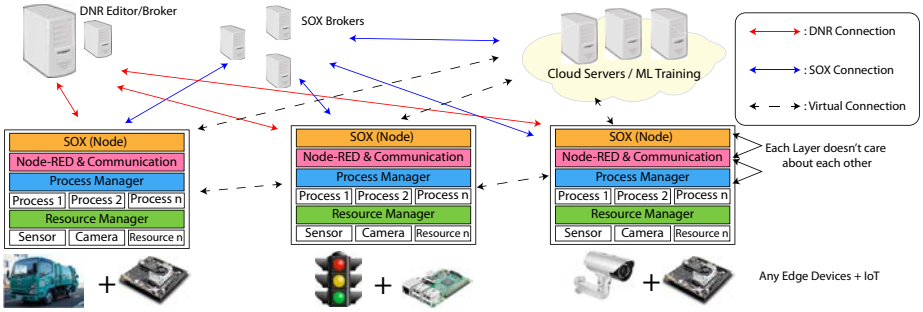


Figure 6.3: Overview of CityFlow.

6.3.1 Design Policy

In order to address the issues described previously, we construct an integrated development environment, called *CityFlow*. This can be used to flexibly realize the life cycle of ML application development in the city. While typical ML applications utilise a centralized processing system, it is difficult to effectively handle the large variety of data sent from the enormous number of edge devices in the city. In contrast, the ML applications, developed within CityFlow, adopt a distributed processing approach, better matched to the spatial-temporal distributed edge environment within a city.

Edge Devices and Preprocessing Virtualization

While the typical ML lifecycle assumes that the data has been already collected into databases, the same does not apply in a city environment. To tackle this, CityFlow collects the data by controlling the edge devices and facilitating the communication between those devices. When collecting the data, it is important to consider the difference between the specification of edge devices and their installed environment. For instance, the data resource could be a virtual sensor such as a web service API or a physical sensors such as an accelerometer. Although the preprocessing should be performed at the sensor where the data is collected, the format of data often differs between platforms, creating additional complexity.

In order to deal with this, CityFlow abstracts away the different edge devices and different data formats to ensure interoperability. Thus, developers are able to concentrate on developing ML models without concern for the underlying technologies and protocols. Furthermore, this abstraction realizes simplified sharing and reusing of these resources among additional developers or applications.

Resource Management and Efficiency Improvement

The task formulation phase and PoC phase in the ML lifecycle is essential to start developing a ML application. In general, it is necessary to select the required data from databases to shift the phase from the task formulation to PoC. However, the presence of data which is not used results in inefficiencies. Therefore, it is efficient for networks and storage mediums to filter the necessary data before storing it into databases. There are several ways to filter data, including:

- **Explicit selection.** Specify the unique device ID to obtain data from it.
- **Implicit selection.** Specify by spatial and temporal range, such as geolocation or time span etc, and collect only data from devices in that range.

The capability of an edge device, such as CPU and memory, is often lower than that used in cloud computing. Consequently, they cannot perform data pre-processing or model prediction in situations where an edge device receives an enormous amount of data. To effectively handle these situations, the device sends the data to multiple nearby edge devices to balance the load.

Domain Adaptation for Spatial-Temporal Covariate Shift

It is necessary to handle the ST covariate shift to develop ML applications in the city scenario described above. However, there are two challenges:

Firstly, since the statistical machine learning including deep neural networks cannot perform extrapolation, we need to take the approach that treats the extrapolated data: ST covariate shift. For instance, we can adopt *domain adaptation learning* which is proposed by Ganin et al. [120]. They assume that a covariate shift occurred between training data (source domain) and test data (target domain). With this assumption, they provide three types of models: the *feature extractor*, which outputs the feature from the common probability distribution wherever the input from the source domain or the target domain, the *solving model* which outputs values for the task (regression or classification), and the *discrimination model* which determines whether the input comes from the source or the target. Although adopting these models is one of the efficient solutions for domain adaptation between two domains, the number of domains in the city might be more than two; it is difficult for the city to adopt these models without change. Therefore, to solve the task, CityFlow is required to distribute the models to edge devices in each domain.

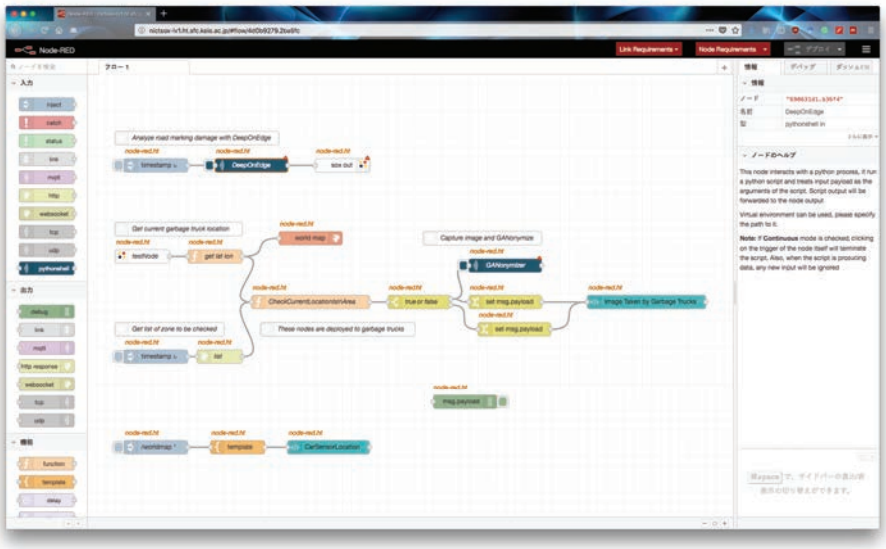


Figure 6.4: Automatic labeling for participatory sensing developed by CityFlow.

The second challenge is the detection of ST covariate shift, which is an estimation of the boundary between the domains as shown in Figure 6.2. In typical ML applications, the meta-information, such as the device IDs and locations, is attached to the data because of being centralized; this is problematic for treating ST covariate shift. Accordingly, CityFlow is required to provide a function that is capable of monitoring the performance of ST covariate shifting and estimating when the boundary occurs. Moreover, CityFlow has to provide the function that can conduct fine-tuning neural networks to alliviate ST covariate shifting.

6.3.2 Implementation

Distributed Node-RED.

Node-RED is a dataflow-based visual programming tool and language for IoT applications. Dataflow is a natural programming model represented as a directed graph of processing nodes, each of which representing a resource. The applications are developed by dragging and dropping processing nodes onto a canvas, and 'wiring' the nodes together. The wires represent communication paths between nodes. The resulting application is referred to as a "flow", which can be deployed to a single device. However, Node-RED has no support for distributed edge environments.

Distributed Node-RED (DNR) is an extension of Node-RED for distributed environments [109, 110]. There are three notable extensions, which are described below.The first extension is the notion of a *device* within the dataflow language. This notion enables individual hardware devices to be uniquely identified in a distributed environment. The device also carries a set of characteristics. This allows the user to determine which device(s) a node should be deployed to and executed on; for example, mobile devices, embedded computers, and cloud servers.

The second extension is the notion of *remote wire*s. *Remote wire*s make it possible to support inter-device communication so that the nodes running on distributed separate devices can send the data. These wires are implemented by using a publish/subscribe communication mechanism. For instance, assuming that node A sends the data to node B. Then node A publishes the data to the communication broker so that node A does not need to know where node B is. Similarly, node B subscribes to the broker so that node B can receive the data without needing to know its source node. Additionally, using *remote wire*s logically separates the process so that highly computational processes can be conducted on the multiple edge devices without using cloud computing.

The last extension is the *constraint* primitive, used as a broader abstraction that specifies how a node is deployed and run in a distributed setting. Accordingly, every node in a dataflow is attached with a *constraint*, property that defines how the deployment is conducted. This *constraint* indicates requirements on device identification, computing resources such as CPU and memory, and physical location. For example, by using the *constraint*, we could restrict a sensor processing flow so this it only runs on a device mounted on a moving vehicle, located within a certain geographic area.

By using DNR for CityFlow, developers can concentrate on developing ML applications without knowing which devices and data resources are used. For more details or other extensions of DNR, see [109].

Sensor-Over-XMPP Node

CityFlow offers a node implemented in DNR that handles Sensor-Over-XMPP (SOX) as shown in Figure 6.6(a). SOX is the specification of SOXFire [121] which is a universal sensor data exchange system. This utilizes the Internet protocol of the open XML format (XMPP), typically used for chat communications, to represent the meta information. Accordingly, by using SOX nodes, the data can be treated uniformly from the physical sensors and the virtual sensors. This feature is useful for the data used in

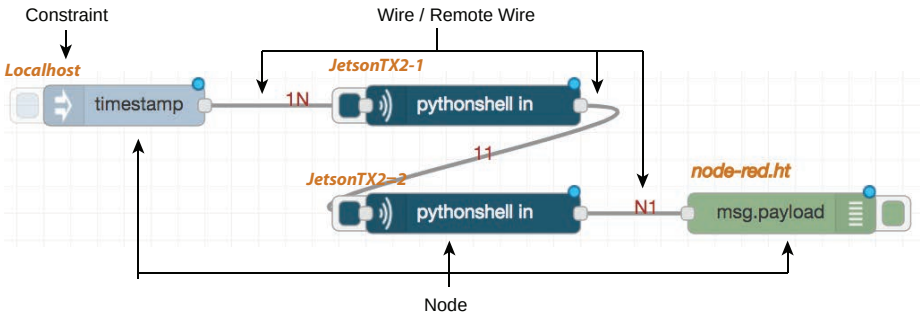


Figure 6.5: Node in Distributed Node - RED, path / remote path, constraints. It is possible to describe the flow of data by connecting multiple nodes by a path, and the deployment conditions of each node by using constraints.

ML techniques in the city. Furthermore, since the ML model is distributed to the edge devices all over a city to conduct prediction locally, SOX can be used.

TensorFlow Node

Simultaneously, we also implement TensorFlow nodes as depicted in Figure 6.6(b). These nodes aim to realize developing an application which deep learning is used for training and predicting. These nodes are implemented by javascript. Though, of course, the functions of these nodes offering such as 'predict' and 'train', are mainly provided as a low-level library by tensorflow-js[3], these functions require users to code lines with expert knowledge. This is not suitable for the goal of CityFlow that it is supposed that CityFlow is used by not only the expert of machine/deep learning but also beginners such as local governments or city officers. In order to achieve the goal of CityFlow, we have to implement the nodes as high level functions to make deep learning procedure simple and easy. For example, in general, developers need to design a neural network architecture, then determine the hyper-parameters for the network (model) optimization. However, it is almost impossible for beginners to do as well. Therefore, we have to mask those procedure from beginners.

According to this strategy, we wrap functions that need to execute a deep learning approach to six kinds of nodes as follows:

- *train* Node.

  This node provides deep learning networks optimization. The hyper-parameters

[3] `https://js.tensorflow.org/`

for optimization, such as 'learning rate', 'batch size' and several optimizer are provided.

- *predict* Node.
  This node is able to use for predicting. Thanks to the function of tensorflow-js, develovers just are required only to prepare the url which indicates the model architecture and its weights. Indeed, to use this node, develovers inputs the url.

- *dataload* Node.
  This node provides the function that loads data as html canvas format so that the data such as images are transformed to array format.

- *img2tensor* Node.
  This node is very important node because most of deep neural networks require their input as tensor and this node tranform the data from array format to tensor format.

- *classify* Node.
  This node is implemented for post-processing. This classify node requires the class labels to tranlate the input vectors to labels, for the input, which is the output of neural networs, represent categorical distribution of classes.

- *dataset* Node.
  The dataset node is used for preprocessing the dataset, such as divide the csv files into data and labels, then split them into train dataset and test dataset.

The example of these nodes usage is denoted in Appendix A.

6.4 Case Study

6.4.1 GeospaceMapping

First, we re-develop our GeospaceMapping by using CityFlow. The result of flow is shown in Figure 6.7. Thanks to the nodes of Node-RED, we can develop our application with only 5 nodes.

- Twitter Node: This node wrap Twitter API so that we do not have to care about the status of Twitter API.

(a) SOX Node. (b) TensorFlow Node.

Figure 6.6: Our custom node which is the component of CityFlow. SOX Nodes contains two kinds of node which are subscribing to SOX and publishing from SOX, respectively. TensorFlow Nodes contains six kinds of node: 'predict' and 'train' node has a function of deep neural networks, 'dataload', 'dataset' and 'img2tensor' are the node for treating data, and 'clasify' is the node for post-processing.

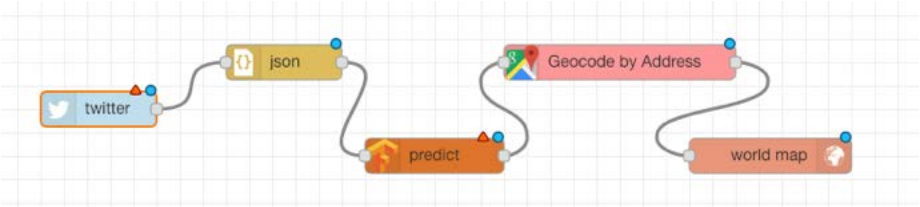


Figure 6.7: Developing GeospaaceMapping via CityFlow.

- json Node: Since the format value of Twitter API is json, we have to parse the json. However, this node enables us not to deal with json format.
- predict Node: This node is our Tensorflow node, thus we only have to do is specifying the URL of the trained neural networks.
- Google Map nodes: The last nodes is embedding the result of location estimation to google maps. Thanks to these nodes as well as Twitter node, we do not need to treat the google map APIs.

Note that though we do not need to engineer Twitter API and Google Map APIs, we have to obtain the API keys of each service respectively.

6.4.2 Road Damage Detection

In this thesis, we use a road damage detection application as a first case study, with using our CityInspector. Although road damage is a common problem, in many cities, in-

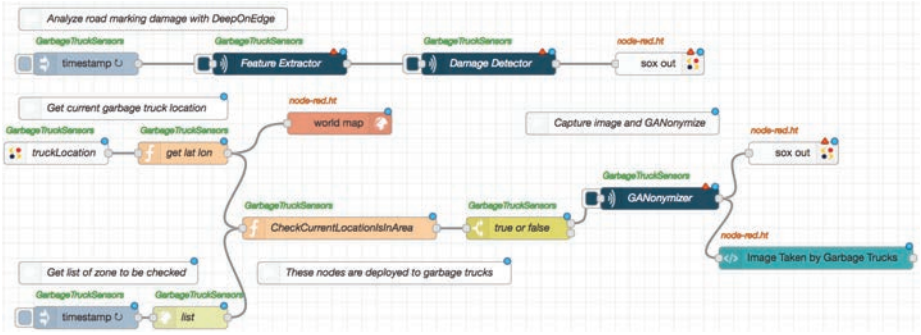


Figure 6.8: Road damage detection application developed by CityFlow.

spection is still conducted by sight. This manual visual inspection is resource intensive and expensive. Therefore, we have to explore ways to inspect the city infrastructures, especially roads, at low cost. In the previous chapter, we proposed a method for road inspection using recorded images from cameras mounted on garbage trucks. However their approach relied on central cloud processing which is inefficient and costly in terms of processing and communications and fails to handle local privacy issues (*Problem 1*). Simultaneously, since we adopted an object detection approach [101] which uses a large neural network, it is difficult to work on edge devices (*Problem 2*). Moreover, the cityscapes of images changes in some areas, so that the ST covariate shift might occur between those areas (*Problem 3*). Our application to address these problems, implemented using CityFlow, operates as follows Figure 6.8:

The flow of Figure 6.8 top.

In order to deploy the networks to edge devices and cope with ST covariate shift, the neural network is separated into *feature extractor* node and *damage detector* node and deployed to different devices on the truck, respectively. Consequently, we can adapt *domain adaptation* approach [120]. Then, when the damaged road is detected through these networks nodes, the location of it is published to SOXFire via SOX Node (sox-out in the flow).

The flow of Figure 6.8 bottom.

When multiple garbage trucks confirm an area of road damage, one truck is designated to upload a partial video of the area. A SOX-in node receives the location information, and the next node compares the location of the truck with it. If it is true, the driving

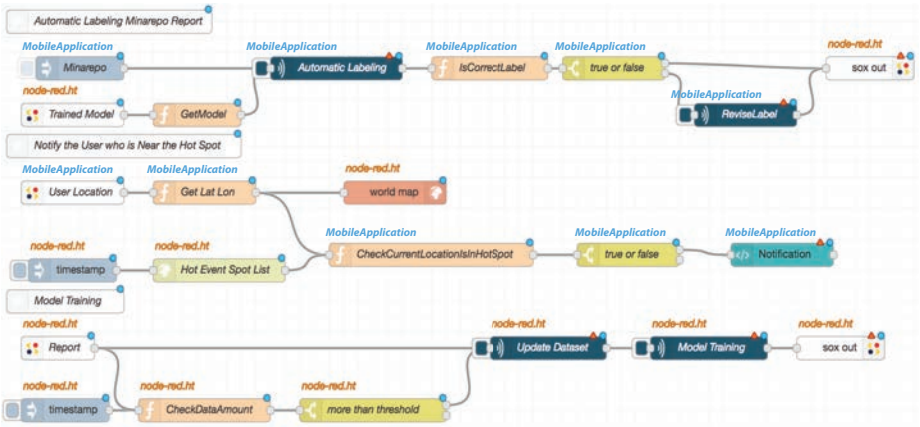


Figure 6.9: Automatic labeling for participatory sensing developed by CityFlow.

recorder mounted on the truck sends the videos after anonymizing to the cloud visualization application. Before uploading, information related to the person such as faces or car matriculation plates is removed if the video contains them. Upload and display the videos in a suitable application for confirmation by city staff.

6.4.3 Automatic Labeling in Participatory Sensing

The second case is the participatory sensing application for city staffs. We assume a system to report when the city staffs discover issues with household garbage such as illegal dumping, uncollected garbage, graffiti etc. Currently, it is used by city staff and the information from reports is gathered daily in several cities in Japan. However, the system requires city staff members to manually input the information about the issue types (12 classes), the degree of urgency (emergency / normal / nothing), and a free comment. This often leads to inconsistencies.

To address this, we have designed an application that automatically labels and makes the report based on image recognition techniques. Given the automatic labeling application, we need to preserve the model quality to be able to label precisely. However, there is no generic ML model (No-free-lunch theorem), so that the model should be selected to suit the area (*Problem 3*) and work on their mobile device (*Problem 2*). In addition, we want to know the situation of specific areas in some cases. To accomplish this, the system requires the city staff members who are in the specific area to check those surroundings (*Problem 1*).

This application flow also can easily be implemented using CityFlow thanks to GUI that can represent the logic intuitively and bind them to devices. The application automatically classifies the photo images taken from the camera by each city staff member. The automatic labeling of the captured image is conducted by the image recognition model. If the result of automatic labeling is incorrect, the city staff member revises it. Then, the information is published by sox out nodes. When the number of reports about the specific area saved to SOXFire exceeds an arbitrary threshold, the system updates the dataset, so that the model performance improves in that area by re-training the model in the cloud servers. It is expected that as more data is gathered, the model performance improves in the specific area.

6.5 Discussion

It is notable that we can develop urban applications which use external service, such as Twitter and Google Maps, without dealing with their APIs owing to Node-RED and SOX. Simultaneously, we does not need to engineer the network configuration thanks to Distributed Node-RED. These characters would make applications development more affordable for not only the experts but also the beginners such as city officers.

However, due to the characteristic of our CityFlow, it is difficult to evaluate it quantatively. Consequently, in this thesis, we demonstrated the our CityFlow can be used to develop our proposed sensing approaches. However, we have to explore that whether our CityFlow can be used for any application development and also it is easy for everyone. For example, we are able to plan to offer the students in the laboratory as experts and the city officers IT division in Fujisawa City as beginners to develop the given application which should be not difficult but important for city administration and check that whether they can be implemented or not.

In future, while the various nodes of Node-Red such as layers of deep neural networks are developed, it will become more flexible and easy to develop the complex application, the prior knowledge about them is also intensively required. For instance, in this thesis, we just assumed that our GeospaceMapping which uses multi-modal data, texts and images, has already been well-designed so that nothing to do for the users who want to develop the application with using it. However, the necessities to develop a new architecture to mix the multimodal data in a different way may occur. Therefore, in that case, we need to provide the nodes that can mix the data flexibly. In

short, the flexibility and simplicity of developing applications by using CityFlow is the trade-off relation and we have to explore the appropriate balance of them.

6.6 Summary

In this chapter, we presented CityFlow, an integrated development environment system for developing urban ML applications which is built using Distributed Node-Red and Sensors over XMPP. We regard the city as the spatial-temporal distributed edge environment where edge devices are installed through the whole city and data is dependant on the time and the edge device location. We have outlined some of the key issues which arise in developing urban ML applications due to the city being a spatial-temporal distributed edge environment. To show how CityFlow is useful, we conducted two case studies for city infrastructure and service improvement. Our implementations highlighted the flexibility in data capture and processing that CityFlow offers and demonstrated that a dataflow model, that virtualizes ML steps as dataflow nodes, makes developing applications easier.

In particular, thanks to Node-RED, we released from engineering the API of web services, and also coding some snippets. This is very affordable for developers not to write similar snippets again and again. Simultaneously, thanks to distributed Node-RED, we can manage the devices very easily. This is also very affordable that developers do not need to consider the situation or context of the devices installed to develop the distributed applications such as our CityInspector.

Chapter 7

Related Work

7.1 Introduction

This chapter introduce the research related to our affordable urban computing. First of all, we describe the studies that have attempted sening not only the citizens but also the phenomenon such as local events from social network services. Then, we explain the works that have attempted to sensing urban environments, such as road condition or air pollution. Finally, we show the studies that have attempted to provide the platform that can deal with distributed computer environments or be used by the users who are not familiar in application development.

7.2 Using SNS to Understand the City

7.2.1 Information Extraction from SNS

One of the easiest methods to estimate the location of tweets posted by users is to find the name of place or spots in the text of a tweet and to match it with geographical corpus. However, this approach is difficult to be applied on abbreviated expressions for specific places, in addition to updating the corpus as a new city or facility appears. Therefore, numerous attempts with machine learning technique have been undertaken. The texts of geotagged tweets are used as training data so that the current places or topics of a local area can be considered. Roller et al. treated this location estimation task as a multi-class classification task, which is explored the same as in this thesis, and classified the geotagged tweets by using bag-of-words (BoW) [61]. For the properties of tweets, the frequency of appearance for each word in a tweet is very low but diverse and the vector in BoW that represents tweets becomes very sparse, which makes the location estimation task more difficult. To address this problem, there are a lot of studies to increase the information of tweets [62][63][64]. However, these approaches required to design the input features and expert knowledge so as to estimate location successfully.

7.2.2 Specific Events Detection

Sakaki et al. [122] detected earthquake from Twitter. They collected tweets which has contained location data or keywords “earthquake” or “shake” or users who have registered a living place in profile, then analysed by Support Vector Machine (SVM). SVM is one of the high-performance machine learning and it can divide into positive

or negative by supervised data. In their research, SVM divided tweets into whether the tweet was related to earthquake or not. Furthermore, it was able to estimate the location close to the epicenter of the earthquake with an accuracy of 96%. Likewise, Lanagan et al. [123] detected characteristic soccer sports events – fouls or goals – by using SVM as well.

7.2.3 Unspecific Events Detection

For previous work to discover the common events, R.Lee et al. [84] focused on the growth rate of *GT-Post*s. They detected events by the rate of *GT-Post*s. As well as this work, Thelwall et al. [124] discovered the events from the amount of *GT-Post*s. Similarly, some works classified the events by tracing the users who post *GT-Post*. R.Lee et al. [84] examined the scale of urban event by tracing the users' mobility in particular region. On the other hand, Haewoon et al. [125], Ishikawa et al. [126], Becker et al. [127] used natural language processing to detect the topics from posts. However, there is limited of notation and information in the technique using natural language processing when they applied to short text such as tweets in Twitter. To cope with this issue, Nishida et al. [128] compressed the posts to acquire the enough amount of information, though the issue still remains when they analyse japanese texts. Also there are previous work that tied urban event and location by Takhteyev et al. [129]. They analysed the relationship among users and make social graph to investigate how events relates to location.

7.3 Urban Environment Sensing

Next, we explain the technologies that utilizing facilities that often exist in the city to capture the environments likewise our proposed CityInspector.

7.3.1 Road Inspection

As well as CityIspector, there are a lot of points to inspect roads with reference to the road inspection. One of those points is flatness. To detect the flatness, the use of accelerometer devices or the smartphones accelerometers is the straightforward approach [92][93][94]. The goal of this study is the road marking blur detection, which is difficult to detect with accelerometers since the value of accelerometers do not change with respect to road marking blur. The other point of roads inspection is cracking. Con-

current to our previous work, Maeda et.al [97] used deep CNNs to detect road damage from images which are uploaded by citizens. They classify the entire image into damaged or not, while our approach detects the blur of several kind road markings at once. Furthermore, they relied on people to give image data, which is called *participatory sensing* [99] and depends on the motivation of participants. While there are numerous works to invent the incentive to make people more likely to participate [130][131], the cost to offer the platform for participatory sensing still remains a problem. In contrast, since we assume using drive recorders that most of garbage trucks are equipped, our system introduction costs may be low.

7.3.2 Using Cameras for Environment Sensing

Using city camera has a great potential to capture the city environment as we demonstrate in Chapter 5. As mentioned in Chapter 3, the cities want to grasp the number of people visits specific place, such as sightseeing spots. In order to count the number of such tourists, while employing people to count it manually is common choise, it is not efficient in the aspect of cost and precision. To tackle this challenge, there is a method that a neural network outputs the number of people who is in the given input image [132]. In this approach, the neural network is trained to output a density map which represents people's spatial location, human body shape and perspective distortion of the given image, and a global number that shows the number of the given image. The training algorithm is quite unique that the neural network minimizes the loss of a density map and a global number not at the same time but alternately.

Another interest technology is proposed by Bak et al. [133], which is person re-identification. Person re-identification is the task to find the same individual across a network of cameras. Bak et al. have adopted deep neural networks to learn a metric using a one-shot learning, which a network learns the task from one or very few training examples. With combining this approach and crowd counting approach, we can track the tourists where they visits all around the city even if they wear different cloths from day to day. This combination can be regarded as the approach in the context of our proposed framework, affordable urban computing.

7.4 Application Development and Management

Finally, we review the research with reference to development and management. There are two aspects in our CityFlow, 1) treating distributed edge computing 2) development and managing environments tools, so-called integretated development environments, to be more comfortable.

7.4.1 Distributed Edge Computing

This thesis presents a novel system called CityFlow, which allows us to build urban ML applications easier. Although there are a number of streaming process engines using multiple devices [113][114][115], they are unable to filter the data according to application spatial or temporal constraint. In contrast, CityFlow can filter explicitly or implicitly, allowing us to collect data flexibly and efficiently, thus better supporting STDEE. While DeepMon [116] or DeepX [117] make it possible to execute deep neural networks on mobile devices by distributing the networks to multiple processes, they only support neural networks. In contrast CityFlow is agnostic with respect ML approach and supports other processes such as data pre-processing and filtering by using DNR. This enables CityFlow to support the complete end-to-end urban application development and, by exploiting DNR's distribution and edge processing capabilities, support a complex ML approach which demands significant processing.

7.4.2 Comfortable Development and Management Tools

One of the development tools that realize the application development without coding is the approaches that convert the images into programming code. For example, pix2code [134] provides the function that with using deep neural networks, input image which is the screenshot of GUI such as HTML, is converted into html source codes. With using pix2code, the efficiency of such html source coding becomes higher than manual coding as usual. Likewise, the tool named MathPix convert screenshots of numerical formula into LaTex codes. This tool relieves us from engineering the structure of the formula.

Similar to our CityFlow, Sony Corporation has develop *Sony Neural Network Console* [135], whose backend is based on *Sony Neural Networks Libarary* (NNL) [136]. NNL is an open source software developed by Sony Corporation, and it is designed as the same concept as other open source frameworks, such as tensorflow [137], py-

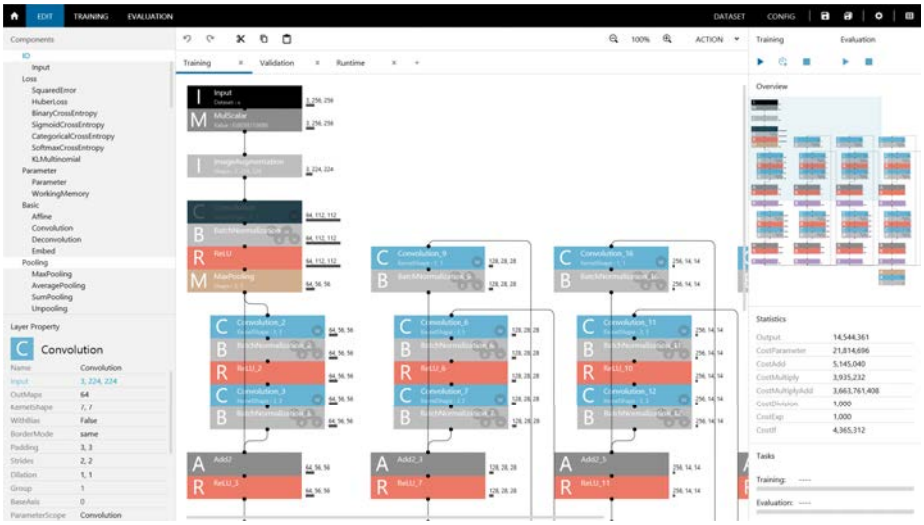


Figure 7.1: Sony Neural Network Console. The image is refered from `https://dl.sony.com/ja/story/`.

torch [138], and chainer [139] etc. Moreover, NNL has performed well in training Imagenet in a large-scale distributed GPU environment [140] and has been used for various products of Sony: gesture recognition of Xperia Ear, synbol recognition of digital paper and so on. Using NNL as the backend, Neural Network Console enables developers to design and implement deep neural networks and conduct evaluation experiments on the graphical user interface as shown in Figure 7.1.

The target of Neural Network Console is the users who regard the efficiency of development as important or the beginners of deep learning, thanks to being supported by intuitive GUI. With this aspect of target users, our CityFlow is similar to the Neural Network Console; the target of CityFlow is the local goverments, such as city officers, who is the beginner of deep learning. Meanwhile, the difference between our CityFlow and Neural Network Console is that our CityFlow is capable of developing applictions to edge devices within the distributed environments. This different point is important to develop deep learning applications in the city. Nevertheless, the function of deep learning supported by Neural Network Console is more substantial than our CityFlow, so that we have to develop our CityFlow continuously.

In addition, we assume data annotation has already done by hand, since preparing the fully-labeled dataset and adopting supervised learning is the best approach with respect to solving the ML task. However, because the cost of annotation has been

Table 7.1: Qualitative Comparition between the relate works introduced in Section 7.3.

item	DSMS[113] D-Stream[114]	DeepMon[]	DeepX[]	NNC[135]	CityFlow
Streming Process	○	×	×	×	○
DL Dev.	–	×	×	○	×
Backend Library	–	Caffe[] DarkNet[]	Lua Torch[]	Neural Networks[136]	TF.js or via Python
Programming Interface	CUI	CUI	CUI	GUI	GUI
Target Device	Common PC	Mobile	Edge	Windows CloudPC	Edge Mobile

highlighted as a significant issue for ML, we have to explore ways to make better use of human annotation [141][142].

7.4.3 Qualitative Comparition

In this section, we show the qualitative comparative results in terms of practicing urban computing as shown in Table 7.1. In this table, we compare the requirements which are needed to realize affordable urban computing. In each cell, '○' means that it is aimed to be satisfied, '×' means that it is aimed to be satisfied but not enough, and '–' means that it is not aimed. Especially, the cell 'DL dev.', that is deep learning development, is that whether validating deep learning performance can be conducted within only the platform or not. As a result, our CityFlow would be suitable for developing urban applications. Though the cell 'DL dev.' of CityFlow is ×, CityFlow can execute python so that if developers need to validate deep learning, they can conduct the validation by python and then they can execute the model within CityFlow.

7.5 Summary

This chapter describes related work of this thesis. Firstly, we reviewed existing research on various association methods. We presented the studies related to human as a sensor, which aims to extract the urban information from the data generated from users, especially social network service. Such as Twitter, is a greatest platform that a lot of claim of citizens is represented so that using and analyzing them is great contribution

for urban computing. However, some studies have used only geo-tagged tweets that we mentioned as much less comparing to whole tweets. Otherwise, the other studies have analyzed not only geo-tagged tweets but also normal tweets to extract some urban information, yet they have not mapped them to geospatial information.

Then, we described the studies that conducting road inspection as same our motivation automating the routine works of city officers. We consider that these studies are very important for realizing smart cities and we need not to exclude but exploit them. For example, to conduct the road inspection, we use dashboard cameras, but we can also use sensors. In short, we have to explore the way to combine these approaches in future. Simultaneously, we explained the studies that using cameras to tackle complicate problems by using deep neural networks. We claim that these studies would contribute to solving the various urban big challenges.

Lastly, we introduced the frameworks that aim to deal with a complicate situation, such as distributed edge environments and deep neural network development. We think that the features of these frameworks is significant factors to develop the cities as much smarter.

The next chapter presents future work on our research and conclude this thesis.

Chapter 8

Conclusion

8.1 Contributions to Urban Computing

This thesis presented the paradigm of affordable urban computing that enables to gather the urban data which is not used in conventional urban computing due to the quality of data or the absence of geospatial information that indicates

The aim of this paradigm is to solve the urban micro challenges which conventional urban computing has not attempted to solve. In order to realize the affordable urban computing, this thesis mentions three problems of requirements: modality translation problem, edge execution problem and development and management problem. To tackle these problems, we showed GeospaceMapping, CityInspector, and CityFlow which are the components of affordable urban computing.

The major contributions of this thesis are as follows:

- Define the problems that occur in practicing urban computing in the current city situation and the requirements of making urban computing able to afford.
- Propose and develop the affordable urban sensing technologies to increase the amount of urban geospatial information and the platform for developing and managing the devices and the generated data in the city.
- Evaluate the effectiveness of affordable urban computing with assuming the real world deployment.

GeospaceMapping enables to transfer the information representing city status to spatial information. Gathering that urban information from users, especially citizens, we can know what happens in a city. On the other hand, due to privacy concerns, there is scarcely to attach the geolocation coordinates to the posts (or tweets). This causes the number of geotagged tweets are posted much less than that of non-geotagged tweets. Though, the information described in non-geotagged tweets is valuable for urban computing. To tackle this issue, we proposed the method to estimate the location that the text and/or images indicate by using deep neural networks. In addition, we demonstrated the two kinds of application that can be deployed when we get a lot of geolocation data. The first application is that estimating the popularity of city events by analyzing the geo-tagged tweets and followees of users who participated in the events and posted about it. The second application is that estimating the transportation modes which citizens used to move around the city by using the features extracted from the sequence of geolocation information by using the variants of recursive autoencoder

which we proposed. With our urban transfer sensing, we can obtain the geolocation information that facilitates us to understand what happens in the city.

Simultaneously, CityInspector enables to detect the road marking blur via object detection approach, which is based on deep convolutional neural networks. We assumed our system is deployed on garbage trucks and works with the images from the drive recorder of garbage trucks. To train our model, we develop a new dataset that contains the images from drive recorders with the annotation with respect to blur; white line, white mark, color line, color mark, crosswalk, guardrails. Our experiments show that the object detection approach performs well at road marking blur detection. By utilizing the trained model, we developed the prototyping of the onboard edge device which equips an embedded CPU, an embedded GPU (so-called NVIDIA Jetson TX2), a USB camera and GPS sensors. To our best knowledge, our CityInspector is the first prototyping that aims to process images in the edge device assumed to work on the garbage trucks and our findings such as the edge devices performance or implementation for power management will contribute to edge devices studies in future.

Lastly, this thesis presents a novel development environments, called CityFlow, an integrated development environment system for developing urban ML applications which are built by combining Distributed Node-Red, Sensors over XMPP and tensorflow. We regard the city as the spatial-temporal distributed edge environment where edge devices which execute processing such as our CityInspector are installed through the whole city and data generated by them is dependant on the time and the edge device location. We have outlined some of the key issues which arise in developing urban ML applications due to the city being a spatial-temporal distributed edge environment. To show how CityFlow is useful, we conducted two case studies for city infrastructure and service improvement. Our implementations highlighted the flexibility in data capture and processing that CityFlow offers and demonstrated that a dataflow model, that virtualizes ML steps as dataflow nodes, makes developing applications easier.

To summarize with the contributions of affordable urban computing, our approaches are the solution to increase the amount of the urban data and to treat the various data within the limited existing resources: texts or images that represent the status of the current city. Since the city is the real physical space, the image sensors (i.e. cameras) is very effective and very generic sensors that capture various status. Simultaneously, the texts generated (written) by people is also the generic sensors to represent various status. With these ‘sensors’ which has already existed in the city, we can exploit them

to understand the city much deeper. However, in the circumstance of the current city, while the data of social network service is available, the video filmed by the urban cameras containing dashboard cameras or surveillance cameras are not gathered due to the lack of environments. In this thesis, we use deep neural networks and edge devices so that we can just start to gather the video data. Additionally, our development environment platform enables us to manipulate these sensors intuitively so that it is more affordable to practice urban computing.

Thanks to deep learning, edge devices, and the integrated Development environments, we can understand the city much deeper and more affordable so that various city not only the big city such as New York City but also the small town such as Fujisawa City can practice urban computing.

8.2 Future Work

This section describes the future work of affordable urban computing. Firstly, we describe future work of technical aspects of affordable urban computing in terms of GeospaceMapping and CityInspector. Then, we describe future work of practicing affordable urban computing related to CityFlow.

8.2.1 Fine Grain GeospaceMapping

Our GeospaceMapping, grain size of the estimated spatial information is only the prefecture of Japan. This is, of course, not enough for understanding the 'city' and we have to explore the way to get more fine-grained geospace, for instance, estimating them with cities level or even in more detail. One of the big challenges to seem to have to be tackled is estimating the geolocation (i.e. latitude and longitude) directly. In order to estimate the geolocation, the straightforward way is using machine learning or deep learning approach in the context of not a classification task but a regression task; the models such as Gaussian process or neural networks output the 'value' indicates latitude and longitude. In the manner of regression tasks, the models are trained to minimize the objective function: mean squared error or mean absolute error. However, this naive objective function is not effective for regressing latitude and longitude. To remedy this issue, one of the hints is the YOLO approach. YOLO model regresses the location (coordinate) of bounding boxes within the given image, though it does not regress the location directly but regress the coordinate relative to the bounds of a grid

cell which is one of the $S \times S$ grid from the image divided. According to this technique, if we can divide the map of Japan to n grid cells, the model would regress the geolocation coordinates. For dividing the map, Google's open source S2 geometry library[1] might be useful. In short, we have to explore the way to map the non-geotagged tweets to the exact location.

Another point of future work in terms of GeospaceMapping is improving the accuracy of itself. GeospaceMapping is the models designed to treat sequential data; text data. Therefore, adopting the recent model architecture, such as transformer [143] which uses attention mechanism [144], would be effective to capture the representation which indicates the place (prefectures).

8.2.2 Smaller, Faster and Low Cost

In terms of our CityInspector, we adopted NVIDIA Jetson TX2 as an edge device in order to execute deep neural networks not on the cloud computers but on the local environment: a local embedded computer. Since the NVIDIA Jetson TX2 is the product and has not only the GPU chip but also the evaluate board, there is no choice that the total cost and the size of CityInspector increased. However, nowadays, there is a movement that developing as much as small devices, so-called *System-on-a-chip* for in the industrial company, such as Intel not to mention NVIDIA. Thus, if the SOC becomes on the market in the future, our CityInspector can be designed and implemented as much cheaper than the current one.

Moreover, our architecture of CityInspector is such generic that any kind of neural networks can be executed. Given the neural network that we used to detect the road marking damage, SSD, although the base network of it SSD is VGG16 which has an enormous number of parameters, it was able to be executed on Jetson TX2. Therefore, in future, our CityInspector will enable to execute other sensing tasks. For example, finding a missing or suspected person from a dash-camera video.

Regarding the software side of executing neural networks on edge devices, one of the key movement is model compression. Such as pruning, distillation, and quantization aim to reduce the size of models in terms of the number of parameters or the data size. For instance, while deep neural networks use 32bit full-precision float points in general, quantization reduces them to half-precision float points or even 1bit float point. If the neural networks are represented by binary float point, the computing through the

[1] https://code.google.com/archive/p/s2-geometry-library/

neural networks can be regarded as electronic signals so that it can be implemented as an electronic circuit; binarized neural networks can be executed on FPGA by much lower power. This technique will accelerate spreading the paradigm of executing deep neural networks on the edge devices.

On the other hand, it is inevitable that the accuracy of those compressed models is lower than that of original models. To remedy this problem, an ensemble method that aggregates the result of models, such as boosting or bootstrap would be effective in the edge environment. Our CityFlow can assign the different process to devices so that we can assign the different models that contain different parameters respectively to the devices to conduct the ensemble method.

8.2.3 Practicing Affordable Urban Computing

CityFlow aims to provide the urban application development environment for not only the experts or developer but also local governments (city officers) and even citizens. The significant future work of CityFlow is to publish CityFlow as open source software. This OSS activity would be accelerated and enhance urban computing more affordable to everyone because one can develop and use the urban computing application in the city. Thanks to the community of Node-RED which is the basic component of CityFlow, the various nodes which posses functions, such as communicating to Twitter API would be developed. This new node development would expand the possibility for beginners such as city officers to develop a novel innovative application. Simultaneously, it can be expected especially the tensorflow nodes to be enriched so as to design and conduct experiments in detail.

With publishing CityFlow over a long period of operation, the amount and the kind of data would be exploded and so that we can understand the long-term change of city, such as the trends of the infrastructure deterioration. Simultaneously, in this thesis, our experiments are aimed only for Fujisawa City. For example, we only use the videos filmed by dashboard cameras of garbage trucks of Fujisawa City. In general, the distribution of data sampled in a different domain (e.g. different light condition) becomes different. Therefore, we assume that the domain bias also occurs not only within the same city but also the different city. In future work, we have to plan to collect the videos of dashboard cameras from other cities such as Samukawa Town to confirm whether the data distribution is biased so that the accuracy of CityInspector decrease or not.

Moreover, the targets of contexts for understanding by computing have shifted: users' surroundings, the status of house, and nowadays the status of a city. These targets shifts have happened because the data is able to be associated with each other to extract much higher abstract knowledge. Indeed, after our affordable urban computing being achieved, the next targets to understand would be several cities at the same time by combining multiple data generated from multiple cities. Consequently, exploring the way to mix the data will be future works.

8.3 Concluding Remark

This thesis has tackled following mainly two issues: 1) an lack of the data for exploiting deep learning technologies to conduct urban computing and 2) the difficulty of the gathered data management and development due to the dynamic spatial-temporal distribution of the devices and the data generated from them. In more detail of the first issue, there are two types of data lacking. Firstly, the data is lacked in terms of partial observation despite the nature of cities being continuous. Secondly, although the data contains plenty of urban geospatial information, the data is not used due to the purpose of the data generated is different from urban computing, for instance, crime prevention. In order to tackle these issues, we proposed affordable urban computing which enables to increase the amount of data within the limited resources. To realize affordable urban computing, we studied and developed three technologies: GeospaceMapping, CityInspector, and CityFlow. With these technologies, we can obtain the data whose quantity and quality are increased and represent the city in more detail so that it can be used to utilize deep learning for urban computing. These works contribute to the field of smart cities and urban computing research and hope to improve the quality of citizens life.

Bibliography

[1] Ioannis Chatzigiannakis. Smart Santander. Technical report, 2011.

[2] City data. https://data.amsterdam.nl/.

[3] DATA-SMART CITY SAPPORO. https://data.pf-sapporo.jp/.

[4] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, Vol. 5, No. 3, p. 38, 2014.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[6] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pp. 247–256. ACM, 2008.

[7] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on gps data for web applications. *ACM Transactions on the Web (TWEB)*, Vol. 4, No. 1, p. 1, 2010.

[8] Robert G Hollands. Will the real smart city please stand up? intelligent, progressive or entrepreneurial? *City*, Vol. 12, No. 3, pp. 303–320, 2008.

[9] Mark Deakin and Husam Al Waer. From intelligent to smart cities. *Intelligent Buildings International*, Vol. 3, No. 3, pp. 140–152, 2011.

[10] Eric Paulos, Ken Anderson, and Anthony Townsend. Ubicomp in the urban frontier. *Human-Computer Interaction Institute*, p. 213, 2004.

[11] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. On discovery of gathering patterns from trajectories. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pp. 242–253. IEEE, 2013.

[12] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252, 2015.

[13] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*, pp. 791–800. ACM, 2009.

[14] Nicos Komninos. *Intelligent Cities: Innovation, Knowledge Systems, and Digital Spaces*. Taylor & Francis, 2002.

[15] Nicos Komninos. *Intelligent cities and globalisation of innovation networks*. Routledge, 2008.

[16] CityVerve. https://cityverve.org.uk/.

[17] San Leandro Next. http://sanleandronext.com/tag/smart-city/.

[18] Smart Shanghai. http://www.smartshanghai.com/.

[19] Smart Nation Singapore. https://www.smartnation.sg/.

[20] Smart Dublin. https://smartdublin.ie/.

[21] ML:Smart. http://www.mksmart.org/.

[22] Barcelona Digital City. https://ajuntament.barcelona.cat/digital/en.

[23] Smart Columbus. https://www.columbus.gov/smartcity/.

[24] Smart Dubai. https://smartdubai.ae/.

[25] The Smart City, City of Stockholm. https://international.stockholm.se/city-development/the-smart-city/.

[26] LinkNYC. https://www.link.nyc/.

[27] Smartsantander. http://www.smartsantander.eu/.

[28] Amsterdam Smart City. https://amsterdamsmartcity.com/.

[29] Mark Weiser. The computer for the 21 st century. *Scientific american*, Vol. 265, No. 3, pp. 94–105, 1991.

[30] Srinivas Devarakonda, Parveen Sevusu, Hongzhang Liu, Ruilin Liu, Liviu Iftode, and Badri Nath. Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*, p. 15. ACM, 2013.

[31] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. U-air: When urban air quality inference meets big data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1436–1444. ACM, 2013.

[32] Rajib Kumar Rana, Chun Tung Chou, Salil S Kanhere, Nirupama Bulusu, and Wen Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 105–116. ACM, 2010.

[33] Fuzheng Zhang, David Wilkie, Yu Zheng, and Xing Xie. Sensing the pulse of urban refueling behavior. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pp. 13–22. ACM, 2013.

[34] Afra Mashhadi, Sourav Bhattacharya, and Fahim Kawsar. Understanding the Impact of Geographical Context on Subjective Well-Being of Urban Citizens. In *Proceedings of the Second International Conference on IoT in Urban Space*, pp. 29–35. ACM, 2016.

[35] Ulf Blanke, Robin Guldener, Sebastian Feese, and Gerhard Tröster. Crowdsourced pedestrian map construction for short-term city-scale events. In *Proceedings of the First International Conference on IoT in Urban Space*, pp. 25–31. ICST (Institute for Computer Sciences, Social-Informatics and ⋯, 2014.

[36] Shinnosuke Wanaka and Kota Tsubouchi. Location history knows what you like: Estimation of user preference from daily location movement. In *Proceedings of the Second International Conference on IoT in Urban Space*, pp. 8–13. ACM, 2016.

[37] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, p. 34. ACM, 2008.

[38] Chih-Chieh Hung, Chih-Wen Chang, and Wen-Chih Peng. Mining trajectory profiles for discovering user communities. In *Proceedings of the 2009 International Workshop on Location Based Social Networks*, pp. 1–8. ACM, 2009.

[39] George Mohler. Marked point process hotspot maps for homicide and gun crime prediction in chicago. *International Journal of Forecasting*, Vol. 30, No. 3, pp. 491–497, 2014.

[40] Alessandro Venerandi, Giovanni Quattrone, and Licia Capra. Guns of Brixton: which London neighborhoods host gang activity? In *Proceedings of the Second International Conference on IoT in Urban Space*, pp. 22–28. ACM, 2016.

[41] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. Urban computing with taxicabs. In *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98. ACM, 2011.

[42] Jing Yuan, Yu Zheng, and Xing Xie. Discovering regions of different functions in a city using human mobility and pois. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 186–194. ACM, 2012.

[43] Der-Horng Lee, Hao Wang, Ruey Long Cheu, and Siew Hoon Teo. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record*, Vol. 1882, No. 1, pp. 193–200, 2004.

[44] Kari Edison Watkins, Brian Ferris, Alan Borning, G Scott Rutherford, and David Layton. Where is my bus? impact of mobile real-time information on the perceived and actual wait time of transit riders. *Transportation Research Part A: Policy and Practice*, Vol. 45, No. 8, pp. 839–848, 2011.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[47] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, Vol. 12, No. Aug, pp. 2493–2537, 2011.

[48] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, Vol. 29, No. 6, pp. 82–97, 2012.

[49] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[52] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, Vol. 529, No. 7587, p. 484, 2016.

[53] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *arXiv preprint*, 2015.

[54] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and

Others. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, Vol. 115, No. 3, pp. 211–252, 2015.

[55] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 843–852. IEEE, 2017.

[56] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pp. 1655–1661, 2017.

[57] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. When will you arrive? estimating travel time based on deep neural networks. AAAI, 2018.

[58] Jeffrey A Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B Srivastava. Participatory sensing. 2006.

[59] Jeffrey Goldman, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, Nithya Ramanathan, Sasank Reddy, Vids Samanta, Mani Srivastava, and Ruth West. Participatory sensing: A citizen-powered approach to illuminating the patterns that shape our world. *Foresight & Governance Project, White Paper*, pp. 1–15, 2009.

[60] Emiliano De Cristofaro and Claudio Soriente. Participatory privacy: Enabling privacy in participatory sensing. *IEEE Network*, Vol. 27, No. 1, pp. 32–36, 2013.

[61] Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1500–1510. Association for Computational Linguistics, 2012.

[62] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pp. 759–768. ACM, 2010.

[63] Yohei Ikawa, Miki Enoki, and Michiaki Tatsubori. Location inference using microblog messages. In *Proceedings of the 21st international conference companion on World Wide Web*, pp. 687–690. ACM, 2012.

[64] Sheila Kinsella, Vanessa Murdock, and Neil O'Hare. I'm eating a sandwich in Glasgow: modeling locations with tweets. In *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, pp. 61–68. ACM, 2011.

[65] Alex Graves. *Supervised sequence labelling*. Springer, 2012.

[66] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, pp. 3276–3284, 2015.

[67] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[68] Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 69–78, 2014.

[69] Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 959–962. ACM, 2015.

[70] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, Vol. 45, No. 11, pp. 2673–2681, 1997.

[71] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

[72] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[73] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[74] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[75] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, 2014.

[76] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, Vol. 18, No. 7, pp. 1527–1554, 2006.

[77] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, Vol. 19, No. 153, 2007.

[78] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, Vol. 11, No. Feb, pp. 625–660, 2010.

[79] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[80] Tao Wang, David J Wu, Adam Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pp. 3304–3308. IEEE, 2012.

[81] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Reading text in the wild with convolutional neural networks. *International Journal of Computer Vision*, Vol. 116, No. 1, pp. 1–20, 2016.

[82] Gergely Palla, Imre Dernyi, Ills Farkas, and Tams Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, Vol. 435, pp. 814–818, 2005.

[83] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, Vol. 446, No. 7136, pp. 664–667, 2007.

[84] Ryong Lee and Kazutoshi Sumiya. Measuring geographical regularities of crowd behaviors for twitter-based geo-social event detection. In *Proceedings of the 2nd ACM SIGSPATIAL international workshop on location based social networks*, pp. 1–10. ACM, 2010.

[85] Ryong Lee, Shoko Wakamiya, and Kazutoshi Sumiya. Discovery of unusual regional social activities using geo-tagged microblogs. *World Wide Web*, Vol. 14, No. 4, pp. 321–349, 2011.

[86] Shinya Hiruta, Takuro Yonezawa, Marko Jurmu, and Hideyuki Tokuda. Detection, classification and visualization of place-triggered geotagged tweets. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pp. 956–963. ACM, 2012.

[87] Hongjun Wang, Hanhuai Shan, and Arindam Banerjee. Bayesian cluster ensembles. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, Vol. 4, No. 1, pp. 54–70, 2011.

[88] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, Vol. 11, No. Dec, pp. 3371–3408, 2010.

[89] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 151–161. Association for Computational Linguistics, 2011.

[90] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pp. 801–809, 2011.

[91] Yu Zheng, Longhao Wang, Ruochi Zhang, Xing Xie, and Wei-Ying Ma. Geolife: Managing and understanding your past life over maps. In *Mobile Data Management, 2008. MDM'08. 9th International Conference on*, pp. 211–212. IEEE, 2008.

[92] Boyu Zhao, Tomonori Nagayama, N Makihata, M Toyoda, M Takahashi, and M Ieiri. IRI Estimation by the Frequency Domain Analysis of Vehicle Dynamic Responses and Its Large-scale Application. In *Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services*, pp. 41–46. ACM, 2016.

[93] Junji Takahashi, Yusuke Kobana, Yoshito Tobe, and Gullaume Lopez. Classification of Steps on Road Surface Using Acceleration Signals. In *proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 229–234. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[94] Tomonori Nagayama, A Miyajima, S Kimura, Y Shimada, and Y Fujino. Road condition evaluation using the vibration response of ordinary vehicles and synchronously recorded movies. *Proceedings of the SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, p. 86923A, 2013.

[95] X Yu and E Salari. Pavement pothole detection and severity measurement using laser imaging. In *Electro/Information Technology (EIT), 2011 IEEE International Conference on*, pp. 1–5. IEEE, 2011.

[96] Kanza Azhar, Fiza Murtaza, Muhammad Haroon Yousaf, and Hafiz Adnan Habib. Computer vision based detection and localization of potholes in asphalt pavement images. In *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on*, pp. 1–5. IEEE, 2016.

[97] Hiroya Maeda, Yoshihide Sekimoto, and Toshikazu Seto. Lightweight road manager: smartphone-based automatic determination of road damage status by deep neural network. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pp. 37–45. ACM, 2016.

[98] Takafumi Kawasaki, Makoto Kawano, Takeshi Iwamoto, Michito Matsumoto, Takuro Yonezawa, Jin Nakazawa, and Hideyuki Tokuda. Damage Detector: The Damage Automatic Detection of Compartment Lines Using A Public Vehicle and A Camera. *EAI MOBIQUITOUS2016IWWSS2016*, pp. ppNA–ppNA, 2016.

[99] Deborah Estrin, K Mani Chandy, R Michael Young, Larry Smarr, Andrew Odlyzko, David Clark, Viviane Reding, Toru Ishida, Sharad Sharma, Vinton G Cerf, and Others. Participatory sensing: applications and architecture [internet predictions]. *IEEE Internet Computing*, Vol. 14, No. 1, pp. 12–42, 2010.

[100] Yin Chen, Jin Nakazawa, Takuro Yonezawa, and Hideyuki Tokuda. Cruisers: An automotive sensing platform for smart cities using door-to-door garbage collecting trucks. *Ad Hoc Networks*, Vol. 85, pp. 32–45, 2019.

[101] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[102] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

[103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[104] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. *arXiv preprint arXiv:1610.02915*, 2016.

[105] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[106] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[107] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, Vol. 20, p. 14, 2015.

[108] M Kawano, K Mikami, S Yokoyama, T Yonezawa, and J Nakazawa. Road marking blur detection with drive recorder. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 4092–4097, dec 2017.

[109] N K Giang, R Lea, and V C M Leung. Exogenous Coordination for Building Fog-Based Cyber Physical Social Computing and Networking Systems. *IEEE Access*, Vol. 6, pp. 31740–31749, 2018.

[110] Nam Ky Giang, Rodger Lea, Michael Blackstock, and Victor C M Leung. Fog at the Edge : Experiences Building an Edge Computing Platform. In *2018 IEEE International Conference on Edge Computing (EDGE)*, 2018.

[111] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[112] Anderson Araújo, Rubem Kalebe, Gustavo Giraõ, Kayo Gonçalves, Bianor Neto, and Others. Reliability analysis of an IoT-based smart parking application for smart cities. In *Big Data (Big Data), 2017 IEEE International Conference on*, pp. 4086–4091. IEEE, 2017.

[113] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, Vol. 15, No. 2, pp. 121–142, 2006.

[114] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. *HotCloud*, Vol. 12, p. 10, 2012.

[115] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, and Others. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 147–156. ACM, 2014.

[116] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 82–95. ACM, 2017.

[117] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, p. 23. IEEE Press, 2016.

[118] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, Vol. 90, No. 2, pp. 227–244, 2000.

[119] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[120] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, Vol. 17, No. 1, pp. 2030–2096, 2016.

[121] Takuro Yonezawa, Tomotaka Ito, Jin Nakazawa, and Hideyuki Tokuda. Soxfire: A universal sensor network system for sharing social big sensor data in smart cities. In *Proceedings of the 2nd International Workshop on Smart*, p. 2. ACM, 2016.

[122] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pp. 851–860. ACM, 2010.

[123] James Lanagan and Alan F Smeaton. Using twitter to detect and tag important events in live sports. *Artificial Intelligence*, Vol. 29, No. 2, pp. 542–545, 2011.

[124] Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. Sentiment in twitter events. *Journal of the American Society for Information Science and Technology*, Vol. 62, No. 2, pp. 406–418, 2011.

[125] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a Social Network or a News Media? In *Proceedings of the 19th international conference on World wide web*, pp. 591–600. ACM, 2010.

[126] Shota Ishikawa, Yutaka Arakawa, Shigeaki Tagashira, and Akira Fukuda. Hot topic detection in local areas using twitter and wikipedia. In *ARCS Workshops (ARCS), 2012*, pp. 1–5. IEEE, 2012.

[127] H. Becker, M. Naaman, and L. Gravano. Beyond trending topics: Real-world event identification on twitter. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[128] Kyosuke Nishida, Ryohei Banno, Ko Fujimura, and Takahide Hoshide. Tweet classification by data compression. In *Proceedings of the 2011 International Workshop on DETecting and Exploiting Cultural diversiTy on the Social Web*, DETECT '11, pp. 29–34, New York, NY, USA, 2011. ACM.

[129] Yuri Takhteyev, Anatoliy Gruzd, and Barry Wellman. Geography of twitter networks. *Social networks*, Vol. 34, No. 1, pp. 73–81, 2012.

[130] Iordanis Koutsopoulos. Optimal incentive-driven design of participatory sensing systems. In *Infocom, 2013 proceedings ieee*, pp. 1402–1410. IEEE, 2013.

[131] Tie Luo, Hwee-Pink Tan, and Lirong Xia. Profit-maximizing incentive for participatory sensing. In *INFOCOM, 2014 Proceedings IEEE*, pp. 127–135. IEEE, 2014.

[132] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 833–841, 2015.

[133] Slawomir Bak and Peter Carr. One-shot metric learning for person re-identification. In *CVPR*, pp. 1571–1580, 2017.

[134] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, p. 3. ACM, 2018.

[135] Sony Corporation. Sony Neural Network Console. https://dl.sony.com/ja/, 2017.

[136] Sony Corporation. Sony Neural Network Library. https://nnabla.org/, 2017.

[137] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and

Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[138] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[139] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[140] Hiroaki Mikami, Hisahiro Suganuma, Yoshiki Tanaka, Yuichi Kageyama, et al. Imagenet/resnet-50 training in 224 seconds. *arXiv preprint arXiv:1811.05233*, 2018.

[141] Hideaki Imamura, Issei Sato, and Masashi Sugiyama. Analysis of Minimax Error Rate for Crowdsourcing and Its Application to Worker Clustering Model. *arXiv preprint arXiv:1802.04551*, 2018.

[142] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya Parameswaran. Helix: accelerating human-in-the-loop machine learning. *Proceedings of the VLDB Endowment*, Vol. 11, No. 12, pp. 1958–1961, 2018.

[143] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[144] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Appendix A

Predicting Flow

```
[
   {
      "id": "38176483.da258c",
      "type": "http in",
      "z": "58dcd408.91cb6c",
      "name": "",
      "url": "/upload",
      "method": "get",
      "upload": false,
      "swaggerDoc": "",
      "x": 350,
      "y": 360,
      "wires": [
         [
            "bcdb0826.17eb18"
         ]
      ]
   },
   {
      "id": "5bc9bf2c.f0b32",
      "type": "http response",
      "z": "58dcd408.91cb6c",
      "name": "",
      "statusCode": "",
      "headers": {},
      "x": 870,
      "y": 360,
      "wires": []
   },
   {
      "id": "bcdb0826.17eb18",
      "type": "template",
      "z": "58dcd408.91cb6c",
      "name": "",
      "field": "payload",
      "fieldType": "msg",
      "format": "handlebars",
      "syntax": "mustache",
      "template": "<!doctype html>\n<html>\n<head>\n<meta charset=\"utf-8\"
          >\n<title>upload</title>\n</head>\n<body>\n<form action=\"/upload
          \" method=\"post\" enctype=\"multipart/form-data\">\n<input name
          =\"file\" type=\"file\" id=\"file\"><br>\n<input type=\"submit\"
          value=\"upload\">\n</form>\n</body>\n</html>",
      "output": "str",
      "x": 540,
      "y": 360,
      "wires": [
```

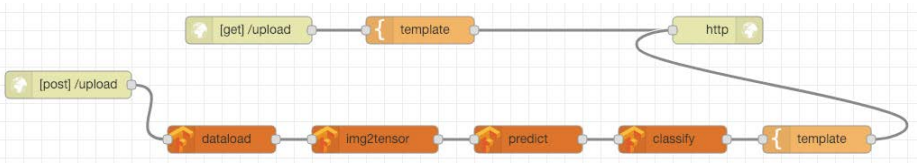


Figure A.1: Predicting Flow.

```
            [
                "5bc9bf2c.f0b32"
            ]
        ]
    },
    {
        "id": "5f447aa9.9d7924",
        "type": "http in",
        "z": "58dcd408.91cb6c",
        "name": "",
        "url": "/upload",
        "method": "post",
        "upload": true,
        "swaggerDoc": "",
        "x": 150,
        "y": 420,
        "wires": [
            [
                "1a79c271.c04cce"
            ]
        ]
    },
    {
        "id": "1a79c271.c04cce",
        "type": "dataload",
        "z": "58dcd408.91cb6c",
        "name": "",
        "x": 320,
        "y": 480,
        "wires": [
            [
                "6f9d480.eaa26b8"
            ]
        ]
    },
    {
        "id": "6f9d480.eaa26b8",
        "type": "img2tensor",
        "z": "58dcd408.91cb6c",
        "name": "",
        "sizew": "224",
        "sizeh": "224",
        "x": 490,
        "y": 480,
        "wires": [
            [
                "4bf752dd.a0aa2c"
            ]
        ]
    },
    {
        "id": "4bf752dd.a0aa2c",
        "type": "predict",
        "z": "58dcd408.91cb6c",
        "name": "",
        "modelurl": "https://storage.googleapis.com/tfjs-models/tfjs/
            mobilenet_v1_0.25_224/model.json",
```

```
        "x": 660,
        "y": 480,
        "wires": [
            [
                "f559199c.2e9a18"
            ]
        ]
    },
    {
        "id": "b6afdcf9.4de7b",
        "type": "template",
        "z": "58dcd408.91cb6c",
        "name": "",
        "field": "payload",
        "fieldType": "msg",
        "format": "handlebars",
        "syntax": "mustache",
        "template": "<!doctype html>\n<html>\n<head>\n<meta charset=\"utf-8\"
            >\n<title>結果</title>\n</head>\n<body>\n<p>{{payload}}</p>\n</
            body>\n</html>",
        "output": "str",
        "x": 980,
        "y": 480,
        "wires": [
            [
                "5bc9bf2c.f0b32"
            ]
        ]
    },
    {
        "id": "f559199c.2e9a18",
        "type": "classify",
        "z": "58dcd408.91cb6c",
        "name": "",
        "classdata": "",
        "x": 820,
        "y": 480,
        "wires": [
            [
                "b6afdcf9.4de7b"
            ]
        ]
    }
]
```

Appendix B

Training Flow

```
[
    {
        "id": "c128d0f8.efce",
        "type": "debug",
        "z": "58dcd408.91cb6c",
        "name": "",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "true",
        "x": 850,
        "y": 680,
        "wires": []
    },
    {
        "id": "483c0149.14081",
        "type": "csv to json ext",
        "z": "58dcd408.91cb6c",
        "name": "",
        "version": "0.1",
        "source": "filename",
        "delimiter": ",",
        "quote": "\"",
        "escape": "\"",
        "ignoreEmpty": false,
        "checkType": false,
        "trim": false,
        "noheader": false,
        "includeColumns": "",
        "headers": "\"f1\",\"f2\",\"f3\",\"f4\",\"label\"",
        "debug": false,
        "x": 370,
        "y": 680,
        "wires": [
            [
                "f598124.ebfadf"
            ]
        ]
    },
    {
        "id": "324c0352.c4817c",
        "type": "inject",
        "z": "58dcd408.91cb6c",
        "name": "",
        "topic": "",
        "payload": "",
```

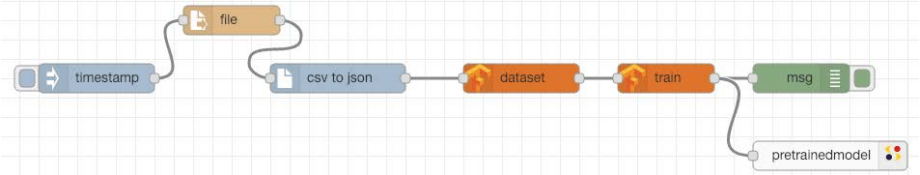


Figure B.1: Training Flow.

```
        "payloadType": "date",
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "x": 120,
        "y": 680,
        "wires": [
            [
                "7b375eb2.039d5"
            ]
        ]
    },
    {
        "id": "f598124.ebfadf",
        "type": "dataset",
        "z": "58dcd408.91cb6c",
        "name": "",
        "dataseturl": "",
        "labelurl": "",
        "x": 560,
        "y": 680,
        "wires": [
            [
                "9a13260f.a15778"
            ]
        ]
    },
    {
        "id": "9a13260f.a15778",
        "type": "train",
        "z": "58dcd408.91cb6c",
        "name": "",
        "lr": "1e-2",
        "batchsize": "128",
        "epochs": "300",
        "x": 710,
        "y": 680,
        "wires": [
            [
                "c128d0f8.efce",
                "59c9e411.f49c7c"
            ]
        ]
    },
    {
        "id": "59c9e411.f49c7c",
        "type": "sox out",
        "z": "58dcd408.91cb6c",
        "name": "",
        "device": "pretrainedmodel",
        "transducer": "",
        "login": "84bf4f56.8cba7",
        "x": 880,
        "y": 760,
        "wires": []
```

```
    },
    {
        "id": "7b375eb2.039d5",
        "type": "file in",
        "z": "58dcd408.91cb6c",
        "name": "",
        "filename": "",
        "format": "utf8",
        "chunk": false,
        "sendError": false,
        "x": 260,
        "y": 620,
        "wires": [
            [
                "483c0149.14081"
            ]
        ]
    },
    {
        "id": "84bf4f56.8cba7",
        "type": "sox-credentials",
        "z": "",
        "nickname": "Default",
        "bosh": "http://sox.ht.sfc.keio.ac.jp:5280/http-bind/",
        "xmpp": "sox.ht.sfc.keio.ac.jp"
    }
    ]
```